UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

SISTEMA EXPERTO PARA LA AYUDA EN LA TOMA DE DECISIONES EN SISTEMAS AÉREOS NO TRIPULADOS (UAS)

Grado en Ingeniería Informática

Guillermo Cerezo Guijarro
02/09/2014

SISTEMA EXPERTO PARA LA AYUDA EN LA TOMA DE DECISIONES EN SISTEMAS AÉREOS NO TRIPULADOS (UAS)

AUTOR: Guillermo Cerezo Guijarro
TUTOR: David Camacho Fernando
Co-Tutor: Gema Bello Orgaz

Grupo de la EPS (opcional) [1]
Dpto. de Ingeniera Informática

Escuela Politcnica Superior
Universidad Autnoma de Madrid
02/09/2014

Resumen

Resumen

El presente trabajo se adentra en un campo que actualmente está experimentando un intenso desarrollo. Se trata de los vehículos no tripulados, comúnmente conocidos como “drones”. Dentro de este campo se pretende crear un planificador de misiones basado en un sistema de restricciones.

Para ello el trabajo está dividido en seis partes: introducción, estado del arte, modelo de datos e implementación, batería de pruebas y resultados, conclusiones y futuras mejoras.

En la introducción se presenta de manera resumida las utilidades que se están dando a los sistemas de vehículos aéreos no tripulados (UAS). A continuación en el estado de arte concentramos el foco en aquellos aspectos de los UAS que se desarrollan en el presente trabajo, es decir los sistemas de guía y los planificadores de misiones mostrando cual es la situación actual de su desarrollo a nivel tecnológico.

Posteriormente creamos el planificador basándolo en restricciones. Para ello se muestran los modelos de datos empleados para el desarrollo del entorno así como el análisis de herramientas basadas en restricciones y las pruebas a las que se las sometieron.

Una vez se ha seleccionado la herramienta sobre restricciones y realizado el modelo, se desarrolla el planificador mediante Gecode (herramienta de CSP seleccionada) y C++. Tras implementarlo se comprueba su funcionalidad así como su rendimiento mediante una batería de pruebas.

El trabajo se cierra con una serie de conclusiones y una sección sobre posible desarrollos futuros del planificador.

Palabras Clave

Vehículo aéreo no tripulado, Sistemas Aéreos no tripulados, Navegación, Control, Guía, Planificador de Misiones, Planificador de Rutas, Automático, Autónomo, Restricción, Alcance, Sensor, Búsqueda en Profundidad

Abstract

The text you are going to read introduces you in a field which is in constant development. We are talking about UAS (unmanned air systems) popularly known as drones. In this text we will develop a mission constraints based based.

The text is divided in six parts: introduction, state of the art, data model and implementation, battery test, conclusions and finally future developments.

In the introduction we present in a brief summary the different functions which UAS are actually receiving. Next, in the state of the art, we focus our attention in those aspects of UAS which are important for our objective, so we describe the current situation of technological development of guide systems and mission planning.

Once we have seen all the theoretical part we develop our planning based in constraints. We build different data models to develop the environment. We analyze different tools based in constraints and we present the results of the different tests they underwent.

After choosing the tool and the model, we developed the planning employing Gecode and C++. Finally we test its functionality and its performance using a set of tests. To close the survey we present the conclusions and the possible future developments of the planning.

Key words

Unmanned Air Vehicle, Unmanned Air System, Navigation, Control, Guidance, Mission Planning, Path Planning, Automatic, Autonomus, Constraint, Scope, Sensor, Depth-First Search

Índice general

Índice de imágenes

Índice de tablas

1 Introducción

1.1. Objetivos y enfoque

Los Unmanned Air System o sistemas aéreos no tripulados, que a partir de ahora los llamaremos UAS, se caracterizan por ser el encargado de la gestión de las misiones de los vehículos no tripulados.

En la actualidad, las posibles aplicaciones de los UAS son muchas y variadas, como puede ser el control y monitorización del estado de los cultivos mediante imágenes multiespectrales, envío de paquetes, rescates, acceder a localizaciones de alto riesgo o la prevención de incendios, entre otros. Durante los últimos veinte años, un gran número de trabajos de investigación en este tema se han llevado a cabo, aplicando diversas técnicas y algoritmos para aumentar las capacidades autónomas de estos.

El sistema de guía dentro de los UAS se podría definir como el "conductor", ejerciendo las funciones de planificación y de toma de decisiones para lograr que las misiones y objetivos que se tengan asignados sean alcanzados. La función de este subsistema es reemplazar los procesos cognitivos de un piloto humano u operador.

Este trabajo trata de abordar una de las partes fundamentales de este subsistema de guía que es el planificador de misiones. Se realizará una misión, que consistirá en visitar unos lugares (objetivos) y las acciones que se tienen que realizar en cada uno de ellos mediante los vehículos (por ejemplo cargar/dejar descarga, adquisición de información, fotografiar, etc...). Los planes que se generen tendrán que tener en cuenta que el entorno es dinámico y cambiante. Utilizando distintos algoritmos de planificación basados en restricciones se deberá generar el sistema de planificación de misiones completo y analizar su eficacia.

1.2. Metodología y plan de trabajo

Durante la evolución del proyecto hemos ido cambiando nuestra filosofía de trabajo dependiendo en que punto del proyecto nos encontrábamos. Comenzamos el desarrollo del mismo recopilando información sobre todo lo relacionado con los sistemas aéreos no tripulados. Una vez leída y analizada iniciamos la creación de nuestro modelo para continuar con la implementación y el realización de las pruebas del mismo. Finalizando con las conclusiones y posibles mejoras en el futuro.

2

Introducción a los UAV

El desarrollo e investigación de algoritmos y mejoras que se pueden aplicar en los sistemas de los vehículos no tripulados, o como se conocen en su forma técnica UAV (unmanned air vehicules) ha estado en continuo avance desde que la informática empezó a tener cabida en todo tipo de vehículos tanto de aire, tierra y mar.

A lo largo de estos meses de trabajo hemos estado investigando como poder realizar un sistema con el cuál permitir al vehículo en cuestión, elegir la forma más sencilla, rápida, y segura de realizar la misión teniendo en cuenta diferentes factores como los elementos del vehículo, así como los distintos puntos de ruta por los que realizará su mision, entre otras muchas características.

Antes de empezar con el proceso de desarrollo del objetivo del proyecto, iniciamos definiendo ciertos términos que tienen que ver con los UAS y que son imprescindibles para la comprensión del proyecto.

2.1. Características de los sistemas UAS

A parte de los UAS, también hay que mencionar los RUAS, rotorcraft UAS, los cuales son los sistemas que podemos localizar en los vehículos con rotor. Ambos sistemas los podemos encontrar en numerosas aplicaciones. El beneficio de la utilización de este tipo de sistemas puede ser muy alta, empezando por evitar la muerte de pilotos en situaciones de rescate de muy alto riesgo, hasta la creación de mapas 3D en territorios prácticamente desconocidos para el ser humano. Se puede entender que un UAS tiene que estar dentro de un vehículo no tripulado, pero no tiene porque ser así, éste puede estar en una estación fija controlando a su vez una cantidad determinada de UAV.

El sistema que tienen a bordo éste tipo de vehículos está formado por tres partes, cada una, con ciertas funciones específicas. Se conoce como GNC (Guidance, navigation and control - Sistema de control de vuelo, sistema de navegación, sistema de guía).

- Sistema de **control** de vuelo: Parte física encargada del control de vuelo, despegue, aterrizaje, así como el control de las distintas partes del vehículo (gasolina, rotor, alas, altitud, etc).
- Sistema de **navegación**: Encargada de realizar el camino más corto, calculando un conjunto de rutas, y entre ellas la más óptima, además de realizar una estimación de posibles caminos alternativos.

- Sistema de **guía**: Se puede definir el sistema de guía, como el "conductor" del vehículo. Es la parte encargada de realizar numerosas funciones entre las que se pueden destacar las siguientes:
  - Razonamiento
  - Toma de decisión a alto nivel
  - Planificiación de la misión y ejecución
  - Planificación de la ruta
  - Decisión de bajo nivel
  - Secuencia de objetivos y generación de la ruta.

Vamos a empezar profundizando en el tema que nos concierne que son los sistemas de guía. Es la parte del sistema donde está integrada el planificador de misiones. Como ya sabemos la principal tarea de un sistema de guía en RUAS/UAS es sustituir/reemplazar el proceso cognitivo de un piloto humano o un operador.

Actualmente la mayoría de los sistemas de guías son realizados mediante operadores remotos (personas desde estaciones en tierra) o sistemas utilizados en los vehículos muy rudimentarios.

Mediante los sistemas de guía se quiere evitar la utilización de operadores humanos, de tal manera que el sistema sea completa o parcialmente autónomo. En el Estado del arte hablaremos más en detalle de los componentes de los sistemas de guía más importantes y que tienen una funcionalidad con mayor grado de importancia o que proporcionan un mayor grado de autonomía según lo acordado en el documento ALFURS[ANEXO A], dicho documento es expuesto por Farid Kendoul en su investigación sobre los GNC[2].

En la imagen 2.1 puede ver un diagrama del sistema GNC, así como las funcionalidades de cada parte y las relaciones entre sí.

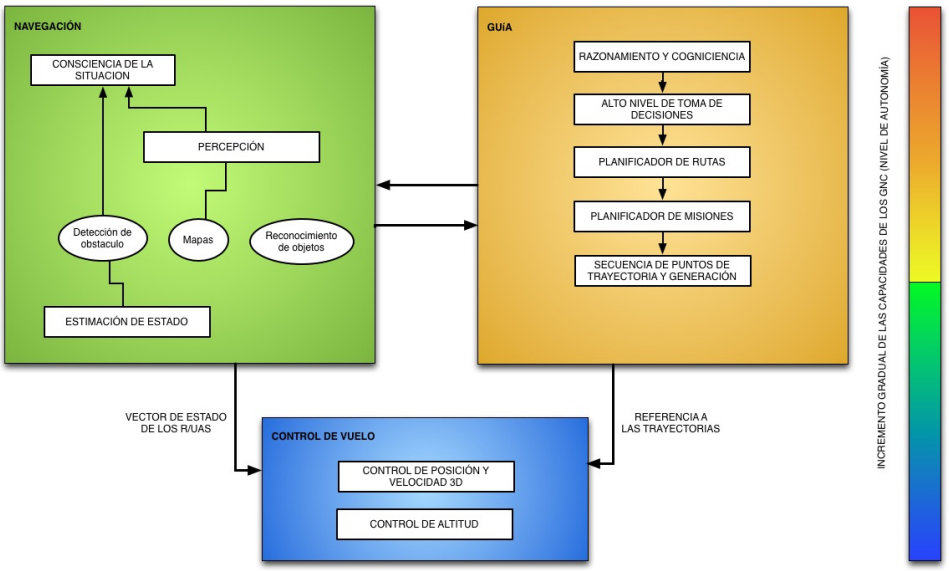


Imagen 2.1: Diagrama del GNC

3

Estado del arte

Dependiendo del nivel de autonomía de los UAS, cada uno de los componentes pueden ser utilizados manualmente por operadores humanos, o automáticamente a bordo del vehículo. Según Clough [3] y Merz [4], es importante conocer las diferencias entre, automático, autónomo y sistema inteligente para poder conocer su nivel de autonomía.

- Un sistema **automático** realizará exactamente lo que está programado para hacer ya que no tiene capacidad de razonamiento o decisión.
- Un sistema **autónomo** debe ser capaz de tomar decisiones y planear sus puntos y ruta/s en función de los objetivos asignados a la misión.
- Un sistema **inteligente** tiene las funciones de un sistema autónomo añadiendo la habilidad de generar sus propios objetivos desde dentro sin tener en cuenta decisiones u ordenes de fuera.

Aunque no se acaban aquí las comparaciones ya que también hay que tener en cuenta otros aspectos para seguir evaluando tanto su autonomía como su rendimiento. Autonomía es un término que esta relacionado con lo que pueden hacer los UAS, mientras que rendimiento está relacionado como de bien realizan la misiones según ciertos requerimientos (tiempo y precisión). La dependencia está relacionado con que frecuencia suelen completar la misión sin problemas (ratio éxito-fallo). De hecho, tanto los diferentes algoritmos existentes, como los sistemas GNC, son desarrollados con el fin de otorgar con el mismo nivel de autonomía posible a los vehículos, teniendo en cuenta los diferentes factores de rendimiento y dependencia. En acorde con un incremento gradual de las capacidades de los GNC [2]-[A] tal como la complejidad de la misión (MC), complejidad del entorno (EC), y la independencia del sistema (ESI), se pueden definir diferentes grados de autonomía para el sistema de guía.

1. Control remoto
2. Control automático de vuelo
3. Eventos adaptados a los RUAS
4. Detección de obstáculos a tiempo real
5. Cooperación a tiempo real y planificador de rutas

6. Planificador de misiones dinámico.
7. Colaboración a tiempo real sobre el planificador de misiones
8. Situación de conciencia y cogniciencia
9. Conocimiento de enjambre y grupo de toma de decisiones
10. Completamente autónomo

Actualmente, existen numerosos sistemas de guía que tienen la capacidad de tratar con diferentes tipos de operaciones sin la intervención de un operador humano. Operaciones como el planificidador de rutas seguida de detección de obstáculos o cambios en la misión. Sin embargo, para llegar a realizar misiones más complejas, o incluso completar misiones coordinadas, son necesarias más capacidades en los sistemas de guía. Estudios e investigaciones actuales están empezando a incluir disciplinas tales como el modelamiento cognitivo, razonamiento e inteligencia artificial dentro de los UAS para proveer un nuevo grado de autonomía en los sistemas de guía, incluyendo claro, planificador de misiones y tomas de decisiones.

A continuación vamos a detallar un poco más tanto el planificidador de rutas como el de misiones.

3.1. Planificador de rutas

Ésta parte del sistema se encarga de acumular la máxima información de vuelo posible que permitirá a los UAS encontrar el camino más rápido y seguro posible para alcanzar alguno de los diferentes objetivos o posiciones.

Es un elemento esencial para alcanzar el nivel cuatro de autonomía. El principal objetivo del planificador de rutas es realizar una ruta entre los diferentes puntos o hitos intermedios/finales evitando cualquier tipo de obstáculo durante el camino. Se pueden identificar seis algoritmos, técnicas o métodos principales que se han utilizado para intentar resolver problemas relacionados con el planificador de rutas:

1. Mapas de carreteras (Road maps - RM): Incluye visibilidad gráfica y rápidos árboles aleatorios de exploración (RRT)[5].
2. Métodos de optimización, incluye progamación entera lineal (MILP), motion primitive automaton (MPA)[6].
3. Algoritmos heurísticos de búsqueda (HSA)[7]
4. Planificación bajo incertidumbre (PPU)[8].
5. Métodos reactivos de evitación de obstáculos bio-inspirados (RBIOAM)[9].

Los planificadores de ruta basados en los métodos del uno al cuatro, se consideran algoritmos que pueden ser globales, locales o una combinación de ambos, mientras que el quinto se considera un método de tipo reactivo. A continuación explicamos las diferencias entre éstos tipos.

Los planificadores de ruta globales generalmente asumen un conocimiento absoluto del mundo y son capaces de proporcionar un camino completo hasta el destino. Estos algoritmos les cuesta realizar un cálculo en tiempo real, es decir, en casos donde hay que revisar o recalcular la ruta cuando nuevos obstáculos son detectados. Pero en su mayoría, muchas de las misiones en UAS no se caracterizan por

tener un conocimiento absoluto del mundo, entendiendo por mundo todos los factores que pueden afectar a nuestro sistema.

Por otro lado los planificadores de ruta locales se pueden definir como un sensor incremental, donde la información es adquirida durante la ejecución de la misión. El mejor camino es determinado basándose en medidas y mapas locales y a diferencia de los planificadores globales, los locales se adaptan mejor a nuevos sensores.

Sin embargo, ninguno de estos algoritmos proporcionan ninguna garantía de seguridad ante eventos inesperados y por lo tanto no garantizan el éxito de la misión. En contraste, los algoritmos de evitación de obstáculos reactivos (RBIOAM) funcionan de manera oportuna y calculan solo la acción siguiente en cada instante, en función del contexto actual. Usan solamente medidas inmediatas del campo de obstáculos para realizar una respuesta reactiva para prevenir colisiones inesperadas o de úlitmo minuto.

Los métodos reactivos son importantes para tratar y resolver obstáculos inesperados de forma rápida y eficaz. Sin embargo, éstos no pueden garantizar una ruta apropiada hasta alguna de las coordenadas parciales o finales.

Una implementación de un planificador de rutas óptimo debería incluir una combinación de los dos algoritmos y métodos hablados en los párrafos anteriores.

3.2. Planificador de misiones

Este componente de los sistemas de guía nos permite aumentar aún más el grado de autonomía del vehículo. El principal objetivo de éste dispositivo está relacionado con la asignación de vehículos para realizar tareas en los objetivos fijados en la misión.

Funcionalmente hablando, dicho componente se sitúa justo por encima del planificador de rutas. El planificador de misiones tiene que generar el plan de misión deseado, es decir, asignar que vehículos van a ir a una serie de puntos que conforman la misión para resolver las tareas especificadas en cada uno de ellos. Hay que aclarar que no es objetivo del planificador de misiones calcular la mejor ruta posible, ni por donde realizarla. Esa tarea concierne a la parte de planificicación de rutas que se explicó en el apartado anterior.

En misiones de vigilancia o supervivencia como se puede comprobar en el articulo de Gianpaolo Conte,[10] ya se ha proporcionado cierta autonomía de alto nivel. También es el caso APEX[11], una arquitectura software proporcionada por la NASA que integra inteligencia artificial capaz de monotorizar eventos, guiar y controlar acciones.

Vamos a detallar un poco más la arquitectura APEX que como podemos ver en la imagen 3.1, se caracteriza por ser el componente inteligente de comportamiento del sistema de guía. Éste, genera planes de misión usando decisiones teóricas aproximadas, basándose en tres capas principales:

1. Capa deliberativa (Planificador de vigilancia)
2. Capa ejecutiva de objetivos (Monotorización y ejecución del plan)
3. Capa de habilidades (piloto automático)

Una vez que ya conocemos los UAS y los diferentes niveles de autonomía que poseen, vamos a hablar del modelo realizado, las herramientas basadas en restricciones investigadas, así como el desarrollo de nuestro planificador.

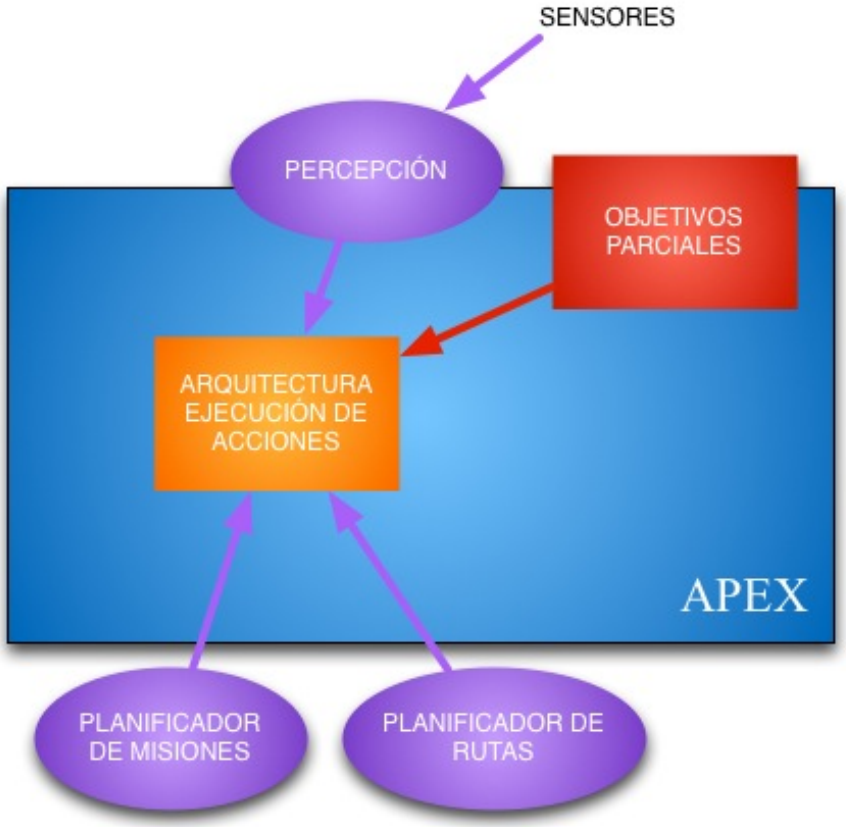


Imagen 3.1: Diagrama de APEX

4

Sistema, diseño y desarrollo

4.1. Sistema desarrollado

Conociendo los elementos de un UAV y el objetivo que deseamos crear, a lo largo de esta sección se va a explicar como hemos ido desarrollando dicho proceso. Con que modelo de datos hemos trabajado, que son los lenguajes de restricciones (CSP), que librerias y programas sobre restricciones hemos probado, así como con cual nos quedamos e implementamos nuestro planificador.

4.1.1. Modelado de misiones de UAVs

Habiendo detallado el planificador de misiones, tendremos que definir nuestro modelo de datos para poder desarrollar nuestro planificador de misiones de UAVs basado en restricciones.

Como ya sabemos el planificador de misiones tiene como objetivo el decidir que UAVs van a ser los que van a realizar las tareas definidas en la misión. En nuestro caso, lo que pretendemos es comprobar que vehículos pueden realizar que tareas y el conjunto de combinaciones posibles.

Por lo tanto, el modelo de datos de nuestro planificador de misiones tendrá que tener en cuenta diferentes factores:

1. Elementos del vehículo
    - Combustible, gasto medio y capacidad del tanque
    - Altitud máxima
    - Velocidad
    - Alcance a la tarea
    - Sensor
    - Peso
2. Sensores del vehículo, se supondrá que el vehículo solo puede contenter tres tipos de sensores:
    - Electroóptico
    - Infrarrojos

- SAR (Synthetic Aperture Radar - Radar de Apertura Sintética)

3. Posición del vehículo, siendo la posición dada por latitud, longitud.
4. Objetivos, o localizaciones que el vehículo tiene que ir para realizar una o varias tareas.

Aunque no solo habrá que tener en cuenta las características del vehículo a la hora de realizar nuestras restricciones, ya que la misión tambien ocupa un lugar importante. Las misiones estarán formadas por un conjunto de tareas, que a su vez constán de los siguientes atributos:

1. Posición: Dada por una longitud y latitud al igual que en los vehículos.
2. Altitud: Toda tarea tendrá una altitud que será a la que se tiene que realizar.
3. Tipo tarea: Se diferenciarán tres tipos de tarea diferentes, de tal manera que cada sensor solo podrá realizar un tipo de tarea.
   - Rescate
   - Reconomiento del terreno
   - Fotografía del terreno

Con estos conocimientos se ha realizado un modelo entidad relación donde exponemos de forma esquemática lo que sería el planificador simplificado, con los atributos más importantes.

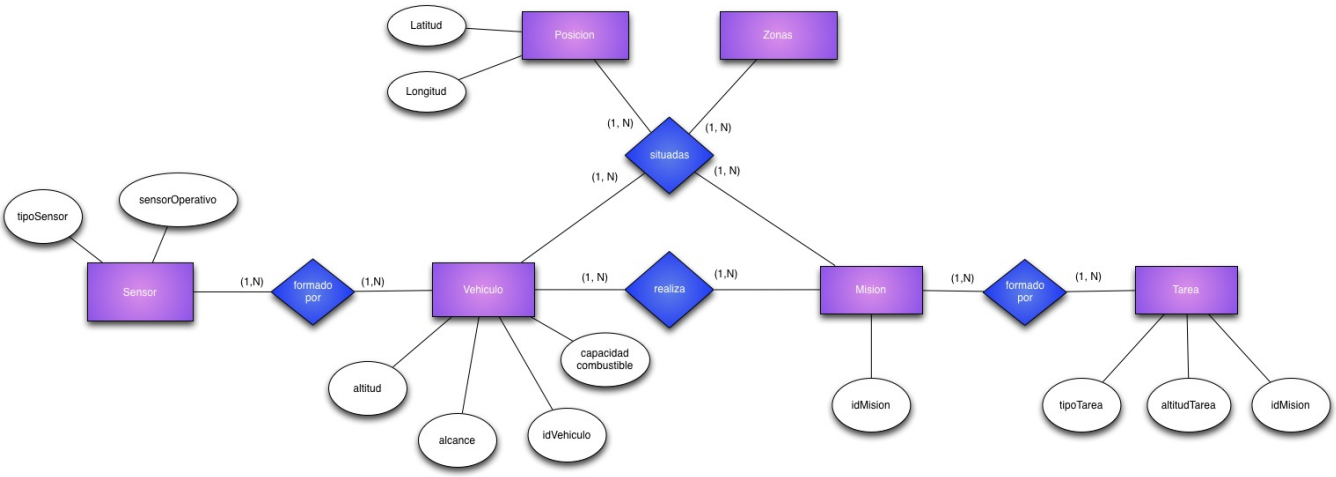


Imagen 4.1: Modelo ER sobre UAVs

En este apartado hablaremos del modelo entidad-relación y de los datos que sacamos del entorno. El modelo definido anteriormente será implementado utilizando ficheros XML para poder trabajar en el desarrollo de nuestro planificador de misiones. A continuación veremos ejemplos de esos ficheros y los comentaremos.

Primer fichero llamado `MisionBD.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
<Mision>
        <idMision>1</idMision>
        <tareaList>
                <Tarea>
```

```
                        <Posicion>
                                <coordenadaX>10</coordenadaX>
                                <coordenadaY>15</coordenadaY>
                                <zonaRestringida>1</zonaRestringida>
                        </Posicion>
                        <altitudMaximaTarea>9000</altitudMaximaTarea>
                        <descripcionTarea>TAREA UNO DE LA MISION</
                           descripcionTarea>
                </Tarea>
        </tareaList>
        <misionFinalizada>0</misionFinalizada>
        <descripcionMision>La mision es la primera y MAS complicada</
           descripcionMision>
</Mision>
</list>
```

Las misiones, como podemos ver están formadas por un número de tareas que puede ser desde uno hasta N, dichas tareas se encuentran en una posición específica dada por una latitud y una longitud. Contendrán además un atributo altitud. Las posiciones pueden encontrarse en estado restringido, aunque comentaremos más adelante como afecta ésto a nuestro planificador.

Segundo fichero llamado `VechiuloBD.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
  <UAV>
    <idVehiculo>1</idVehiculo>
    <tiempoVuelo>30</tiempoVuelo>
    <peso>4.0</peso>
    <alcance>50000</alcance>
    <combustible>5</combustible>
    <altitud>100</altitud>
    <velocidad_max>30</velocidad_max>
    <velocidad_min>70</velocidad_min>
    <sensorList>
      <Sensor>
        <tipoSensor>SAR</tipoSensor>
        <zoom>30</zoom>
        <operativo>1</operativo>
        <alcance>50</alcance>
      </Sensor>
    </sensorList>
    <Posicion>
      <coordenadaX>30</coordenadaX>
      <coordenadaY>50</coordenadaY>
      <zonaRestringida>0</zonaRestringida>
    </Posicion>
  </UAV>
</list>
```

Como podemos ver, consideramos que el vehículo esta formado en todo momento por como mínimo: un sensor y se encuentra en una posicion inicial dada por una latitud y una longitud. La capacidad del tanque de gasolina se considerará que son litros y se supondrá que con todos los litros del tanque se podrá recorrer el número de kilómetros introducidos como alcance del vehículo.

Tanto el alcance, como la altitud se consideran que están en kilómetros. En general la altitud se suele medir en pies, aunque en nuestro caso esto será en km, como referencia se tiene que un UAV de alta

altitud se encuentra entre 10.000-30.000 pies, y que en conversión serían 3.000-9.000 km[12].

Tercer fichero llamado `SensorBD.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
  <Sensor>
    <tipoSensor>Electrooptico</tipoSensor>
    <zoom>100</zoom>
    <operativo>1</operativo>
    <alcance>100</alcance>
  </Sensor>
  <Sensor>
    <tipoSensor>Infrarrojos</tipoSensor>
    <zoom>20</zoom>
    <operativo>1</operativo>
    <alcance>20</alcance>
  </Sensor>
  <Sensor>
    <tipoSensor>SAR</tipoSensor>
    <zoom>30</zoom>
    <operativo>1</operativo>
    <alcance>50</alcance>
  </Sensor>
</list>
```

Los tres sensores anteriores serán los que utilizaremos a lo largo de todo el proyecto y donde cada uno será capaz de realizar solo un tipo específico de tarea. Hablaremos de ello en el apartado 4.2.3.

Dicho modelo lo hemos realizado basándonos en los conocimientos que hemos sacado a lo largo de toda la información obtenida, especialmente el artículo de Kendoul[2] y la conferencia de Merz sobre sistemas autónomos[4].

Para implementar el parseador de objetos se ha utlizado la libreria <libxml> de c++ que nos ha permitido crear un parseador que leerá todos los ficheros con la información detallada anteriormente.

4.2. Algoritmos sobre el planificador de misiones

Como se ha podido leer en el punto uno, nuestro objetivo era crear un planificador de misiones modelándolo como un problema de satisfacción de restricciones, la tarea de dicho planificador es asignar tareas a nuestros UAVs para que éstos las realicen. De esta manera, se eligió implementarlo a través de satisfacción de restricciones (CSP) ya que a la hora de asignar tareas a vehículos se aplicarían las restrcciones que se estipulen, evitando así asignaciones innecesarias que dicho vehículo no podría realizar. En las siguientes líneas se presentará una introducción a este tipo de problemas.

4.2.1. Problemas de satisfacción de restricciones (CSP)

Según Delisle y Schwartz[1] la programación por restricciones es una metodología software utilizada para la descripción y posterior resolución efectiva de cierto tipo de problemas, típicamente combinatorios y de optimización. Estos problemas aparecen en muy diversas áreas, incluyendo inteligencia artificial, investigación operativa, bases de datos, sistemas de recuperación de la información, etc. La programación de restricciones puede dividirse en dos ramas claramente diferenciadas: la “satisfacción de restricciones” y la “resolución de restricciones”. Ambas comparten la misma terminología, pero sus

orígenes y técnicas de resolución son diferentes. La satisfacción de restricciones trata con problemas que tienen dominios finitos, mientras que la resolución de restricciones está orientada principalmente a problemas sobre dominios infinitos o dominios más complejos. Comenzaremos resolviendo este tipo de problemas mediante un modelo, este proceso tiene unos sencillos pasos:

- Definir las variables (finitas en nuestro caso).
- Crear un dominio de valores finito para cada variable.
- Definir un conjunto de restricciones que acotan las combinaciones válidas de valores que pueden llegar a tomar las variables.
- Resolverlo mediante un método de búsqueda.

De esta manera ya tendríamos el problema definido, como dice en el último paso, para resolverlo se utilizan técnicas de búsqueda de la solución, apoyadas generalmente por criterios heurísticos. El objetivo es encontrar un valor para cada variable del problema de manera que se satisfagan todas las restricciones del problema. En general, la obtención de soluciones en un CSP es NP-completo, mientras que la obtención de soluciones optimizadas es NP- duro, no existiendo forma de verificar la optimalidad de la solución en tiempo polinomial.

Informalmente, una restricción es una secuencia de variables junto con las combinaciones de valores permitidas para dicha secuencia. Las tecnologías basadas en restricciones calculan soluciones a problemas de restricciones razonando en torno a las restricciones que definen el problema, las variables sobre las cuales las restricciones han sido definidas, y los dominios finitos (es decir, los posibles valores) de estas variables.

4.2.2. Estudio de herramientas para resolver CSPs

Existen diversas herramientas de software libre que permiten resolver CSPs. Para elegir una adecuada a nuestro sistema se ha realizado un estudio. A continuación en la imagen 4.2 podemos encontrar una extensa y detallada tabla con los diferentes programas encontrados, sus limitaciones y plataformas[13].

PROGRAMAS	Sistema Operativo	Descripción	Instalación	Lenguaje
AISpace	Linux/Windows/Mac	Esta herramienta es utilizada para aprender y explorar conceptos de inteligencia artificial	Sencilla, ya que un .jar que no necesita de instalación	Realizas una clase de modelo de datos, no es un lenguaje en especial
MINION	Linux/Windows/Mac	Herramienta basada en resultados empíricos sobre los puntos de referencia estándar que muestra órdenes de magnitud más mejoras de rendimiento	Hay que instalar un conjunto de librerías para poder empezar a trabajar	
GECODE	Linux/Windows/Mac	Herramienta encargada de resolver restricciones basándose en las librerías proporcionadas por Gecode	Hace falta instalar un programa en MacOSx y Windows. Linux con aplicar el MakeFile que te proporcionan es suficiente	C++ pero con GECODE utiliza su propia librería para ciertos aspectos, como la definición de variables
Elisa	Linux	Elisa es una librería abierta de C++ para resolver problemas de restricciones	Es necesario instalar GOAL con sus respectivas librerías para poder trabajar con C++	C++
HSolver	Linux	HSolver es un programa para la verificación de los sistemas híbridos basados en el solucionador de restricciones RSOLVER	Primero hace falta instalar RSolver, con sus respectivas librerías necesarias	RSolver
Choco	Linux, Windows, Mac	Choco es una librería java creada para resolver problemas de restricciones basado en eventos de propagación		Java
ECLIPSe		Eclipse es un sistema de software de código abierto para el desarrollo y el despliegue de aplicaciones de programación de restricciones	Herramienta externa	Código abierto

Imagen 4.2: Aplicaciones CSP probadas

Las conclusiones que se pueden obtener de los programas analizados en la tabla es que la mayoría todavía se encuentran en desarrollo y en la mayoría su API o documentación es mínima. De entre todas cabe destacar las tres siguientes:

- AISpace: Aplicación con interfaz gráfica, muy fácil de usar y sencilla que te permite muy bien entender problemas de este tipo, aunque no se puede utilizar para casos de gran complejidad.
- Elisa: Librería abierta de c++ que te permite trabajar con csp, solo funcionará con la aplicación Goal, y la api y la información disponible es muy escasa.
- Gecode: Al igual que Elisa, es una herramienta para el desarrollo de sistemas basados en restricciones, es el programa elegido para desarrollar nuestro algoritmo y del cual hablaremos en el siguiente apartado.

4.2.3. Implementación sobre Gecode

El porque de la utilización de Gecode es sencillo, de todo el conjunto de programas probados es el más completo. Te permite trabajar en todo el abanico de sistemas operativos y tiene una API extensa y detallada. Para entender correctamente nuestro modelo de restricciones vamos a explicar con un ejemplo sencillo el funcionamiento tanto de gecode, como de un problema de restricciones. Vamos a realizar el problema de las n reinas, el cual definimos a continuación: Colocar en un tablero de dimensiones 8x8, ocho reinas de manera que ninguna de ellas ataque a otra bajo las reglas del ajedrez. De esta manera, vamos a presentar el código realizado en Gecode y explicar las partes de un problema de restricciones definidas anteriormente, éste código es proporcionado por Gecode a través de su página de ejemplos[14].

Fichero llamado `nueveReinas.cpp`:

```
#include <gecode/int.hh>
using namespace Gecode;
class Reinas : public Space {
protected:
IntVarArrayreina;
public:
Reinas() {
const int n = 8;
reina = IntVarArray(*this, n, 0, n - 1);

        distinct(*this, reina, ICL_VAL);
         IntArgs c(n);
        for(inti=n;i--;)c[i]=i;
                distinct(*this, c, reina, ICL_VAL);
        for(inti=n;i--;)c[i]=-i;
                distinct(*this, c, reina, ICL_VAL);
        branch(*this, reina, INT_VAR_SIZE_MIN, INT_VAL_MIN);
}
Reinas(bool share, Reinas& s) : Space(share,s){
        reina.update(*this, share, s.reina);
}
virtual Space* copy(bool share) {
        return new Reinas(share,*this);
}
// Funcion para imprimir la solucion
};
// Funcion principal
```

- Definición y creación del dominio de variables
  Para implementar el modelo, por simplicidad, definimos una constante de tipo entero n para especificar el número de reinas, y un vector reina de n variables de decisión de tipo entero. Para usar variables de decisión y restricciones de tipo entero, es necesario incluir <gecode/int.hh>. El constructor del vector de variables de decisión de tipo entero toma como primer argumento el espacio actual. (De hecho, cualquier función que dependa del espacio, toma al espacio actual como argumento y por tanto esto se repite en los constructores de variables de decisión, funciones que establecen restricciones, y funciones que especifican ramificadores.)

- Definición del conjunto de restricciones
  Podemos definir como restricción la siguiente regla que llamaremos *todosDiferentes*, definida sobre una secuencia $\{x_1, ..., x_n\}$ de n > 0 variables tipo entero, que se cumple que si y solamente si cada par de variables entera x_i y x_j toma valores diferentes para i j (con i, j $\in [1, n]$).

  Para lograr expresar las restricciones *todosDiferentes(n)* en las diagonales del tablero, en el constructor, declaramos un vector de argumentos de tipo entero IntArgs(Línea 12 de nueveReinas.cpp). La memoria asignada a los vectores de variables de decisión (como IntVarArray) se libera solamente cuando termina la vida del espacio en el que han sido definidos. Esto hace que este tipo de vectores no sean adecuados para variables temporales, en este caso para ser usados como argumentos en la especificación de restricciones. Los vectores de tipo IntArgs, al contrario, son inmutables, obtienen espacio del heap, y la memoria es liberada cuando su destructor es ejecutado.

  La implementación de la restricción *todosDiferentes(n)* en Gecode es llamada restricción distinct. Establecer distinct(space, x) (Línea 11, 14 de nueveReinas.cpp) exige que todas las variables en x tomen valores diferentes. Éstas dos funciones pueden tomar un último parámetro especificando el nivel de consistencia para la reducción por propagación. En este caso, especificamos el nivel de consistencia por valores a través de la constante ICL_VAL indicando que la reducción por propagación se efectuará cuando alguna variable sea asignada. Gecode te permite definir tus propias relaciones mediante el métdo rel, además en la programación con restricciones, éstas se implementan a través de propagadores.

  El siguiente paso es especificar un ramificador que, usualmente, toma un vector de variables que deben ser asignadas durante la búsqueda, junto con una estrategia de selección de variables y una estrategia de selección de valores. En nuestro caso (Línea 17 de nueveReinas.cpp), usando una heurística de fallar pronto, especificamos para el vector de variables reinas, seleccionar primero las variables con el dominio de menor cardinalidad (a través de la constante INT_VAR_SIZE_MIN) y luego asignar primero el menor valor del dominio (a través de la constante INT_VAL_SIZE_MIN).

  Por la manera en que la búsqueda es implementada en Gecode, es necesario implementar un constructor de copia y una función de copia (Línea 19, 22 de nueveReinas.cpp). La función de copia es virtual para que sea posible crear una copia de un espacio aún cuando no se conozca la subclase exacta de este. El argumento share no debería preocupar al lector puesto que es usado internamente por Gecode.

- Finalmente, para encontrar la solución, además de una función para imprimir los valores asignados a nuestras variables de decisión es necesario implementar la función principal donde se realizará la creación de un motor de búsqueda en profundidad DFS.

Ya conocidos como funciona Gecode y para poder entender bien el modelo de restricciones habrá que explicar el modelo unificado, que podemos ver en la imagen 4.3, detallando los métodos creados para aplicar las restricciones, así como las diferentes clases implementadas.

Comentados los detalles de los atributos de cada clase a través de los ficheros XML, conviene empezar a hablar sobre los métodos implementados para poder realizar las restricciones. Uno de ellos calcula

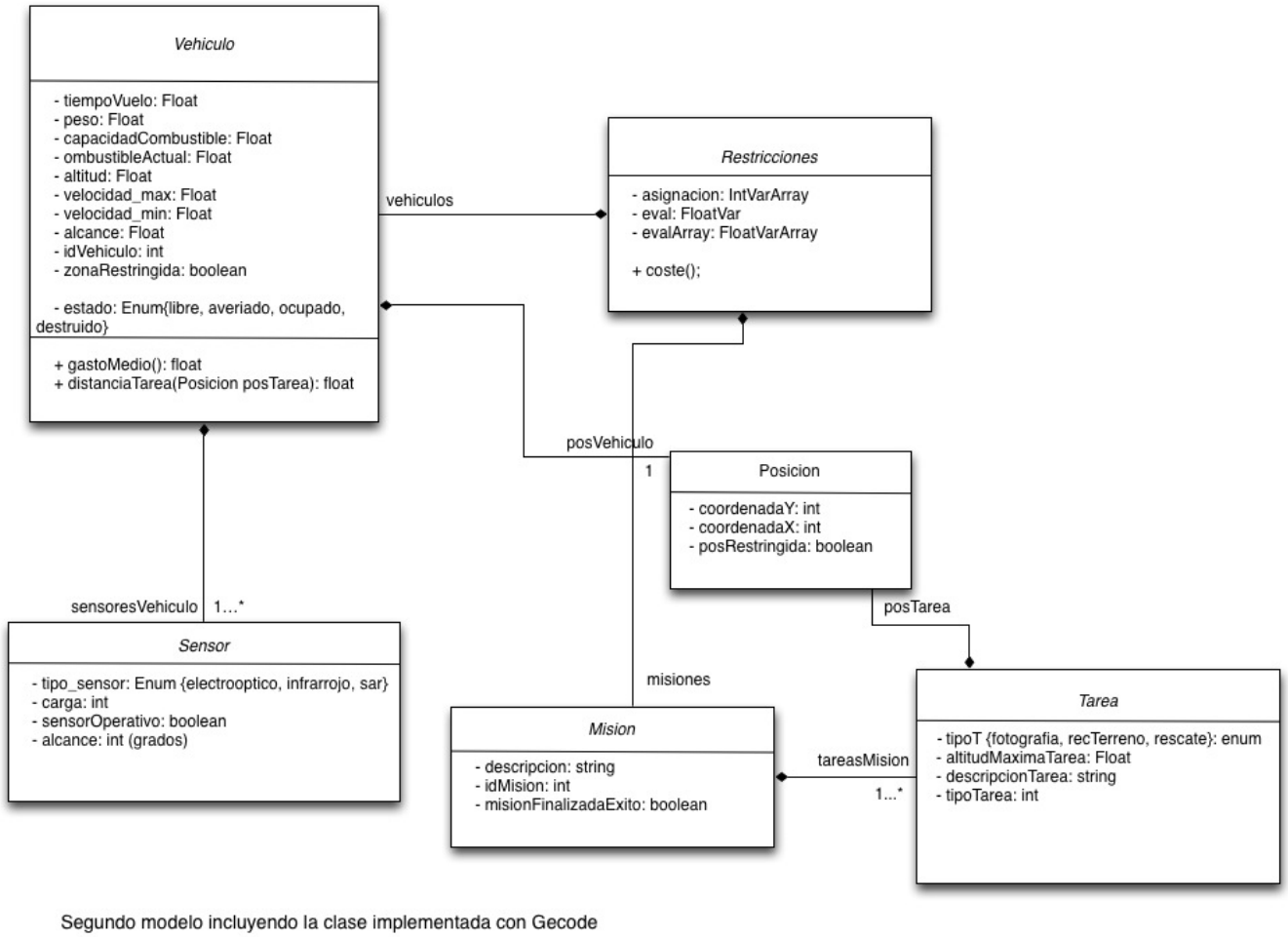


Imagen 4.3: UML del planificador

las distancias entre vehículos y tareas, ya que inicialmente se tienen sus posiciones específicas, las cuales pueden ir desde -90 hasta 90 grados (-90/+90, -90/+90). Las distancias entre ellos se calculan aplicando la siguiente regla matemática con la que obtenemos la distancia entre dos puntos del mapa, de ésta manera:

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Por otro lado el consumo de gasolina es otro detalle importante a comentar. Inicialmente se tiene la capacidad del tanque de gasolina, así como el alcance del vehículo. Se supondrá que el UAV irá a una velocidad media constante, por lo que su consumo medio se calculará mediante la división de los litros de capacidad y el alcance. El consumo de gasolina irá directamente relacionado con la distancia, ya que cuanto mayor es la distancia a recorrer en una tarea, mayor cantidad de gasolina se consumirá del vehículo.

También habrá que comentar el concepto de posición restringida. En el caso de que una posición contenga el espacio aéreo restringido activo (su valor será true), quiere decir que ningún vehículo podrá realizar dicha tarea, excepto que el vehículo posea permisos para entrar en la posición y pueda realizarla. Nos referiremos a éstos permisos como si el valor booleano del objeto estará a true también. Mediante un parseador convertiremos los datos captados de la base datos en objetos para poder crear un planificador en nuestro entorno. Ahora vamos a explicar las restricciones que tenemos y que utilizaramoes en la clase Restricciones.cpp para evitar asignaciones innecesarias:

- Altitud de la tarea: La altitud máxima del vehículo deberá superar la altitud de la tarea ya que

en caso contrario querrá decir que el vehículo no es capaz de alcanzar la altitud del objetivo. Se resolverá comparando una con la otra.

- Sensor operativo: El sensor del vehículo deberá estar operativo durante la misión para poder realizar la/s tarea/s.
- Sensor vehículo-Tipo Tarea: El tipo de sensor tiene que coincidir con el tipo de tarea, de tal manera que con un sensor de infrarrojos no se podrá realizar una tarea de rescate. Los sensores siguientes podrán realizar los siguientes tipos de tareas:
  1. SAR - Rescate
  2. Infrarrojos - Reconocimiento terreno
  3. Electroóptico - Fotografía del terreno
- Posición restringida: Si una posición se caracteriza por tener el espacio aéreo restringido, los vehículos podrán entrar solo si tienen posRestringida a true.
- Gasolina: Se calculará el gasto medio del vehiculo hasta la tarea y se comprobará que dicho gasto no supera la gasolina total del vehículo.
- Alcance: Se calculará las distancia que hay entre el vehículo y la tarea y se comprobará que ésta no es mayor al alcance del vehículo.

¿Cómo crearemos nuestro planificador? Una vez convertidos los datos de XML a nuestros objetos de tipo Vehiculo, Tarea, etc, nos encargaremos de definir las asociaciones entre estos. Para ello, utilizaremos Gecode, a continuación la implementación de nuestro código.

1. Definición de variables, crear un dominio de valores e implementación de restricciones
   Tendremos un conjunto de vehículos y tareas los cuales queremos relacionar entre sí. De esta manera, la idea es crear un array de tipo entero del tamaño del número de tareas donde iremos asignando los vehículos. Por lo tanto, si tenemos por ejemplo cuatro tareas y tres vehículos, un ejemplo de ejecución sería el siguiente:

   - Primera iteración: EL vehiculo con id=1, realiza las cuatro primeras tareas, tal que quedaría, 1-1-1-1.
   - Segunda iteración: EL vehiculo con id=1, realiza las tres primeras tareas y el vehículo con id=2 la última, tal que, 1-1-1-2.
   - Tercera iteración: EL vehiculo con id=1, realiza las dos primeras tareas y la última mientras que el vehículo con id=2 la tercera. La salida sería, 1-1-2-1.
   - Cuarta iteración: EL vehiculo con id=1, realiza las dos primeras tareas, el vehículo con id=2 la dos últimas, tal que quedaría, 1-1-2-2.

   Y asi sucesivamente, hasta realizar todas las combinaciones. De esta manera definiremos por un lado un array de tipo enteros para guardar los vehículos que van a realizar las tareas y cuyo tamaño será del número de tareas. El dominio de éste será de 0 al número de vehículos. Por otro lado un array del mismo tamaño del anterior de tipo bool que guardará un valor de 0 o 1 dependiendo si el vehículo cumple con los requisitos y podrá realizar la misión. A continuación la definición de nuestras variables y el bucle con el que realizaremos las asignación:

   Fichero llamado `Restricciones.cpp`:

```
aignacion = IntVarArray(*this, NUMTAREAS, 0, NUMVEHICULOS-1);
restriccion = BoolVarArray(*this, NUMTAREAS, 0, 1);

for(int j=0; j<NUMTAREAS; j++){
        for(int i=0; i<NUMV; i++){
                rel(*this, asignacion[j], IRT_EQ, i, restriccion[i]);
                }
        }
}
```

El array restricción será a su vez una relación creada por nosotros donde realizaremos las restricciones. A continuación las restricciones implementadas:

```
rel(*this, restriccion[i], IRT\_EQ, vehiculo[i].getAltitud()<=mision.
    getTareas[j].getAltitudTarea());
rel(*this, restriccion[i], IRT\_EQ, mision.getTareas[j].
    getPosicionTarea().getZonaRestringida()==vehiculo[i].
    getZonaRestringida());
rel(*this, restriccion[i], IRT\_EQ, vehiculo[i].getSensorVehiculo().
    getSensorOperativo()>=fin);
rel(*this, restriccion[i], IRT\_EQ,  vehiculo[i].getAlcance()<=
    vehiculo[i].distanciaTarea(mision.getTareas[j].getPosicionTarea()))
rel(*this, restriccion[i], IRT\_EQ, vehiculo[i].gastoMedioCombustible
    ()*(vehiculo[i].distanciaTarea(mision.getTareas[j].getPosicionTarea
    ()))
```

Aunque arriba podrá ver un conjunto de relaciones en nuestro caso se resumirá todas las condiciones en un único rel, uniendo las condiciones mediante and o &&, pero se ha dividido para que se entienda mejor.

De esta manera la relación principal cumpliría la siguiente regla de Gecode. rel(home, x, IRT_EQ, y, r); De tal manera que si r es asignado a uno, la restricción x=y se propaga.

Un propagador es una función reducción que lleva a cabo reducción por propagación. Por tanto, los propagadores son funciones que se aplican a un conjunto de los dominios de las variables y básicamente lo que hacen es reducir los dominios de estas variables eliminando los valores que no pueden ser asignados de acuerdo con la restricción que implementan.

Una vez realizado las restricciones, se definirá dicho propagador o branching que tomará como vector de variables el de *asignacion* y la estrategia a seguir mediante INT_VAL_SPLIT_MIN() será que elija el siguiente valor más pequeño del vector. El ramificador que utilizamos sería:

branch(*this, asignacion, INT_VAR_NONE(), INT_VAL_SPLIT_MIN());

Ya definidos el ramificador y las restricciones solo queda resolver las asignaciones.

2. Resolverlo mediante método de búsqueda. Mediante Gecode podemos utilizar dos algoritmos de búsqueda para resolverlo:

   - DFS (Depth first search o búsqueda en profundidad): Es un motor que utiliza la búsqueda secuencial para encontrar la solución. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.
   - BAB (Branch and bound - Ramificación y poda): Éste motor encuentra las soluciones y las imprime, siendo la mejor solución la que se imprima la última. Utiliza la técnica de

ramificación y poda, es decir, todas las soluciones posibles se disponen en un árbol y el algoritmo de resolución va descartando las soluciones menos óptimas.

En el último paso de nuestro código realizaremos una búsqueda en profundidad dentro del main de tal manera que nuestro código será el siguiente:

```
DFS<Restricciones> e(restricciones);
        delete restricciones;
while (Restricciones* s = e.next()) {
        s->print(f);
        delete s;
}
```

El código enseñado es una pequeña parte del código de Restricciones.cpp, además se han implementado todas las demás clases del modelo con sus respetivcas variables y métodos. En la siguiente sección comenzaremos la realización de las pruebas del planificador, realizando previamente una función de medida de calidad de soluciones.

5

Experimentación

5.1. Experimentación sobre el modelo

5.1.1. Descripción del conjunto de datos

Partiendo del modelo visto, podemos comenzar a hablar de los diferentes vehículos y tareas que vamos a utilizar para realizar nuestro conjunto de pruebas. La cantidad de ambos no será muy alta ya que al utilizar búsqueda en profundidad se puede llegar a producir una cantidad muy alta de soluciones, extenderemos este razonamiento más en profundidad en el apartado de resultados.

Nuestro objetivo es tener un conjunto de tareas y vehículos con diferentes características entre sí. Siendo éste nuestro punto inicial, empezaremos hablando de las misiones que tendrán que realizar los vehículos. Definiremos en total dos misiones y cinco tareas por cada una.

Se crearan un conjunto de tareas con características completamente diversas, tanto en altitud, como posición y tipo. La principales diferencias entre la misión uno y la dos será que se encontrarán en posiciones y altitudes bien alejadas unas tareas de otras, y por otro lado que las tareas de la misión uno solo serán de un único tipo, mientras que las de la misión dos podrán ser de cualquier tipo. A continuación en las tablas 5.1 y 5.2 los atributos de cada tarea de las dos misiones.

Tareas	Altitud	Posición	Tipo de tarea
T1	3000	-50, 20	Rescate
T2	3400	-60, 25	Rescate
T3	3800	-70, 30	Rescate
T4	4000	-80, 35	Rescate
T5	4500	-90, 40	Rescate

Tabla 5.1: Misión Uno

Las tareas de está misión tendrán un altitud ”baja” y su tipo de tarea será solo de rescate.

Tareas	Altitud	Posición	Tipo de tarea
T6	7000	30, -22	Fotografía
T7	7200	40, 54	Reconocimiento terreno
T8	7400	50, -27	SAR
T9	8200	0, 44	Reconocimiento terreno
T10	9000	70, 7	Fotografía

Tabla 5.2: Misión Dos

Las altitudes de las tareas de está misión serán más altas, llegando a hasta un límite de 9000 km y su tipo de tarea variará.

En el caso de los vehículos, crearemos nueve vehículos en total para nuestreas pruebas. Vamos a separar dichos vehículos en tres grupos diferentes para su explicación y poder ver las similitudes y diferencias entre sí, en las tablas solo aparecerán los atributos que pueden afectar a las restricciones.

- Los tres vehículos del primer grupo se caracterizarán por tener un atributo que no cumpla una de las restricciones y de esta manera no podrán completar ninguna de las misiones.
- El segundo grupo contendrá vehículos que tendrán uno de los atributos aplicados a la función de coste óptimo o cercano a lo que se puede considerar el caso óptimo.
- El tercer grupo estará formado por vehículos en los cuales todas sus características se acercarán al caso más óptimo y por lo tanto todos ellos podrán realizar el conjunto de tareas explicado anteriormente.

Vehículos	Altitud máxima	Posición	Capacidad Combustible	Alcance	Número de Sensores	Tipo de Sensor/es	Sensor operativo
V1	500	35, 20	1000	1500	1	SAR	Sí
V2	6000	50, 15	100	1500	1	Electroóptico	No
V3	6500	80, 12	600	200	1	Infrarrojos	Sí

Tabla 5.3: Grupo uno de vehículos

En la tabla 5.3, el primer vehículo tiene una altitud demasiado baja para realizar ninguna de las tareas. El segundo UAV no tiene el sensor operativo y el tercer caso tiene un consumo medio de gasolina tan alto que no será capaz de llegar a ninguna tarea.

Vehículos	Altitud máxima	Posición	Capacidad Combustible	Alcance	Número de Sensores	Tipo de Sensor/es	Sensor operativo
V4	7000	20, 20	100	800	3	SAR, Electro-óptico, Infrarrojos	Sí
V5	8500	13, 11	200	250	1	Electroóptico	Sí
V6	4500	30, 10	100	1300	2	SAR, Infrarrojos	Sí

Tabla 5.4: Grupo dos de vehículos

En este segundo caso, el vehículo de la tabla 5.4 con id 4, tendrá la característica de que tendrá todos los tipos de sensores y podrá realizar cualquier tipo de tarea. El segundo caso tendrá una altitud muy alta que le permitirá realizar casi todo tipo de tareas a cualquier altitud. El último, tendrá un consumo muy bajo de gasolina.

Vehículos	Altitud máxima	Posición	Capacidad Combustible	Alcance	Número de Sensores	Tipo de Sensor/es	Sensor operativo
V7	8000	0, 42	450	2000	3	SAR, Electro-óptico, Infrarrojos	Sí
V8	8500	-35, 18	200	3000	3	SAR, Electro-óptico, Infrarrojos	Sí
V9	10000	89, 77	156	2000	3	SAR, Electro-óptico, Infrarrojos	Sí

Tabla 5.5: Grupo tres de vehículos

Por último en la tabla 5.5, los tres vehiculos, tendrán todos los tipos de sensores, una altitud máxima muy alta y un consumo de gasolina realmente bajo.

A la hora de realizar pruebas trabajaremos con estos nueve vehículos principalmente aunque para alguna prueba específica de rendimiento crearemos vehículos/tareas nuevos pero con características similares a los anteriores.

5.1.2. Medidas de calidad de soluciones

Es importante realizar una selección correcta de los atributos que serán los encargados de realizar la medida de la calidad de las soluciones. De esta manera, habrá que partir de la base de las restricciones que se realizan, y en función de éstas decidir cual se le da un mayor o menor grado de importancia. Una vez conocidos los factores, normalizarlos para obtener un dato entre 0-1 y por lo tanto tener un valor más específico para analizar mejor la salida. En las salidas obtenidas, uno será el resultado del conjunto de vehículos cuya salida es la mejor para realizar una misión y cero el peor caso.

La utilización de muchos factores para realizar dicha función de medida no es bueno ya que, se acabaría dando muy poca importancia a todos los factores sin llegar a una conclusión clara. En este caso, lo que se pretende a partir de estos factores es beneficiar a los vehículos cuyos factores cumplan con las condiciones que consideramos óptimas. Basándonos en las restricciones anteriores, éstos son los factores elegidos y que consideramos sus respectivas restricciones como más importantes:

- Distancia hasta la tarea (Alcance): Este factor es realmente importante ya que un vehículo que éste más cerca que otro de una tarea debería tener mejor puntuación sobre el otro. Se tendrá en cuenta si un vehículo debe ir a más de una tarea.
- Gasto medio de combustible: Este atributo va directamente conectado con el anterior, ya que un vehículo que este más lejos que otro, pero cuyo gasto de gasolina sea inferior y sea capaz de llegar a la tarea habiendo consumido menos gasolina a pesar de recorrer más distancia, habrá que puntuarlo de manera favorable.

- Altitud máxima: A primera vista la altitud puede parecer un atributo poco reseñable, pero conviene destacar su importancia, ya que un vehículo cuya altitud es baja o más bien "poco alta", no será capaz de realizar todo tipo de misiones. En consecuencia, habrá que puntuar favorablemente la mayor altitud de unos vehículos sobre otros.

Una vez conocidos los factores que vamos a utilizar para realizar dicha función, deberemos normalizarlos para obtener nuestra salida de datos.

1. ¿Cómo normalizar dichos factores?

   Lo que se pretende es puntuar favorablemente a aquellos vehículos que cuyos factores se asemejen con lo dicho anteriormente. Para normalizar dichos atributos se búscara el mejor caso posible y se normalizara a partir de este valor. De esta manera:

Altitud	Gasto combustible
$altitudVehculo/altMaximaVehculo$	$gastoMnimoVehculo/gastoVehculo$

Tabla 5.6: Normalización de la altitud y el gasto medio de combustible

Se comprobará cual es la altitud mayor de los vehículos y las tareas, y está será la utilizada como mejor caso. En el caso del gasto de combustible, se hará de la misma manera y el vehículo cuyo gasto de combustible sea el menor, este será el mejor caso.

La distancia no se incluye en la tabla anterior ya que es un caso especial porque, cuál sería el mejor caso de la distancia entre un vehículo y una tarea? En el que la distancia fuera cero y por lo tanto es imposible normalizar con el mejor caso. Debido a esto se utlizará el peor caso, se comprobará cual es el vehículo que se encuentra a mayor distancia y se realizará la siguiente normalización:

$$(peorDistancia - distanciaVehiculoATarea)/peorDistancia$$

De esta manera un vehículo cuya distancia hasta la tarea sea de cero, obtendrá una puntuación de uno. En los tres casos, se pretende puntuar alto en general de tal manera que un vehículo que tenga atributos cercanos al que se consideré nivel óptimo deberá obtener una puntación cercana a uno. Además hay que destacar que en el caso de la distancia se tendrá en cuenta la distancia recorrida por un mismo vehículo hacia varias tareas.

Dichos factores normalizados se calcularán por cada salida, aplicándose a cada vehículo con su respectiva tarea. Llegados a este punto ya podemos comenzar a realizar la batería de pruebas para comprobar el funcionamiento del planificador.

5.1.3. Resultados experimentales

A partir de los vehículos y tareas creados anteriormente comenzaremos a realizar la batería de pruebas. Primero realizaremos un análisis de la salida en función de nuestras pruebas, relacionando nuestros vehículos con nuestras misiones, aumentando poco a poco la cantidad de vehículos-tareas y presentando a través de una tabla el conjunto de mejores resultados de nuestras asignaciones. El caso de la mejor asignación se mostrará en color azul dicha fila.

Segundo, realizaremos pruebas computacionales, para comprobar cual es el rendimiento de la búsqueda DFS.

Para referinos a cada grupo de vehículos, lo nombraremos con G1, G2, G3. En el caso de las misiones serán M1, M2.

1. G1 realiza M1 || G1 realiza M2
   En este primer caso realizaremos dos pruebas en uno ya que su resultado será idéntico. El conjunto de vehículo de G1 se caracteriza por cumplir cada uno una restricción de tal manera que nunca podrán realizar ninguna de las misiones.

2. G2 realiza M1
   En este caso, no todos los vehículos podrán realizar la misión ya que el vehículo con id 5, no tendrá el sensor SAR que le permite realizar una misión de rescate. Por lo demás, tanto V4, como V6 podrá alcanzar todas las tareas tanto en altitud, como en los demás requisitos. Se obtendrá en consecuencia, dos vehículos que podrán realizar las cinco tareas de forma combinada, esto nos daría un total de 32 soluciones, a comprobar en la tabla 5.7:

Número	Solución	Altitud	Distancia	Combustible
1	0.415182	0.875	0.667577	0.533333
2	0.433476	0.8125	0.754878	0.6
3	0.433486	0.8125	0.754932	0.6
4	0.440459	0.75	0.78563	0.666667
5	0.432941	0.8125	0.752205	0.6
6	0.441046	0.75	0.788563	0.666667
7	0.441057	0.75	0.788616	0.666667
8	0.437841	0.6875	0.768369	0.733333
9	0.431859	0.8125	0.746793	0.6
10	0.441082	0.75	0.788743	0.666667
11	0.441093	0.75	0.788797	0.666667
12	0.438995	0.6875	0.774143	0.733333
13	0.440547	0.75	0.78607	0.666667
14	0.439582	0.6875	0.777076	0.733333
15	0.439593	0.6875	0.77713	0.733333
16	0.427306	0.625	0.711531	0.8
17	0.430271	0.8125	0.738854	0.6
18	0.440593	0.75	0.786301	0.666667
19	0.440604	0.75	0.786355	0.666667
20	0.439606	0.6875	0.777197	0.733333
21	0.440059	0.75	0.783628	0.666667
22	0.440193	0.6875	0.78013	0.733333
23	0.440204	0.6875	0.780184	0.733333
24	0.429017	0.625	0.720083	0.8
25	0.438976	0.75	0.778216	0.666667
26	0.440229	0.6875	0.780311	0.733333
27	0.44024	0.6875	0.780365	0.733333
28	0.430171	0.625	0.725856	0.8
29	0.439694	0.6875	0.777638	0.733333
30	0.430758	0.625	0.728789	0.8
31	0.430769	0.625	0.728843	0.8
32	0.410511	0.5625	0.623389	0.866667

Tabla 5.7: Soluciones Prueba 2

Como podemos ver, la solución once es la más óptima, ¿porque de éste resultado?. Primero sa-

ber cual es la solución once, en esta tenemos el siguiente orden donde cada vehículo realiza las siguientes tareas:

Asignación
V4 realiza T1
V6 realiza T2
V4 realiza T3
V6 realiza T4
V4 realiza T5

Tabla 5.8: Asignaciones de la mejor solución - Prueba2

Sabemos que el V4 se caracteriza por estar a mayor altitud que V6, pero éste se caracteriza por tener un consumo muy bajo de gasolina. Eso se puede comprobar como en el primer y el último caso, son los valores más altos de salida en altitud y gasto de combustible respectivamente. El caso mejor se caracteriza porque las distancias recorridas por los dos vehículos son más bajas que los demás casos y su gasto de gasolina es muy bajo también.

3. G2 realiza M2
   Inicialmente está misión no podrá realizarla ningún vehículo de está tarea debido a que ningún vehículo alcanza la altitud mínima de la tarea 10. Por eso se van a modificar las alitudes de los tres vehículo para ver como se comportaría en ese caso, y cuales serían las posibles soluciones. De esta manera las altitudes quedarían:

Vehículos	Altitud Prueba 2	Altitud Prueba 3
V7	7000	8200
V8	8500	9000
V9	4500	7000

Tabla 5.9: Nuevas altitudes para los vehículos

A pesar de las nuevas altitudes, éste conjunto de vehículos con esta misión tendrá las siguientes restricciones:

- V5 solo podrá realizar las tareas 6 y 10 debido a su tipo de sensor.
- V6 debido a su altitud más baja solo podría realizar la tarea 6, pero con ninguno de sus senores puede realizar esa tarea.
- V4 gracias a tener los tres tipos de sensores y una alta altitud podrá realizar todas las tareas menos la 10.

Cumpliéndose todas éstas restricciones, tendremos solo dos soluciones posibles en la tabla 5.10.

Número	Solución	Altitud	Distancia	Combustible
1	0.400345	0.928889	0.629501	0.443333
2	0.40076	0.946667	0.703802	0.353333

Tabla 5.10: Soluciones Prueba 3

Asignación Uno	Asignacion Dos
V4 realiza T5	V5 realiza T6
V4 realiza T6	V4 realiza T7
V4 realiza T7	V4 realiza T8
V4 realiza T8	V4 realiza T9
V5 realiza T10	V5 realiza T10

Tabla 5.11: Asignaciones de la mejor solución - Prueba3

En la tabla 5.11 podemos ver las asignaciones entre vehículos-tareas y se obtienen dos soluciones las cuales son muy parecidas entre sí. El segundo caso, se sabe que los vehículos recorren menos distancia y de ahí que su valor sea mejor, pero por otro lado al consumir el vehículo cinco considerablemente más que el cuatro, la solución uno tendrá mejor valor de combustible. Al ser las dos soluciones tan pareciadas, se podrían considerar cualquiera de las dos como la mejor.

4. G3 realiza M1
   Éste grupo de vehículos se caracteriza por tener todos los factores muy óptimos, pudiendo realizar todas las tareas y por consiguiente que el valor de salida final sea mayor en general. De esta manera, al tener tres vehículos que pueden realizar cinco tareas, habrá un total de 243 soluciones posibles, ya que imprimir esa cantidad de soluciones lo encontrabamos inviable, se mostrarán las 20 mejores soluciones.

Número	Solución	Altitud	Distancia	Combustible
122	0.548528	0.85	0.892641	1
123	0.548844	0.88	0.893279	0.97094
125	0.548724	0.88	0.89268	0.97094
131	0.548056	0.88	0.889341	0.97094
149	0.546846	0.88	0.883292	0.97094
203	0.545126	0.88	0.874689	0.97094

Tabla 5.12: Soluciones Prueba 4

Como se puede apreciar en la tabla 5.12, la solución 123 es la mejor, en la cuál se han producido la siguientes asignaciones:

Asignación
V8 realiza T1
V8 realiza T2
V8 realiza T3
V8 realiza T4
V9 realiza T5

Tabla 5.13: Asignaciones de la mejor solución - Prueba4

Con respecto a las soluciones obtenidas, V8 es el vehículo más óptimo de los tres ya que es el más cercano a todos esos puntos, tiene una altitud muy alta, y el menor consumo de todos los vehículos que pueden realizar dicha misión, de ahí que ese caso sea el mejor, aunque tampoco dista tanto de los demás ya que también son vehículos con atributos muy óptimos. La única diferencia del mejor caso con las demás soluciones es que su factor de distancia es un poco mejor al de los demás.

5. G3 realiza M2
   A continuación G2 realizará el conjunto de tareas de la misión dos, a diferencia del caso anterior,

esta vez no todos los vehículos pueden realizar todas las tareas. Solo V9 que cumplirá todos los requísitos podrá realizar todas las tareas.

Los otros dos vehículos no alcanzan la altitud ni de la tarea diez, ni de la tarea nueve y solo podrán realizar las tarea seis, siete y ocho. Por lo que las posibles soluciones serán 54, ya que tenemos tres vehículos que pueden realizar las tres primeras combinaciones (total de 27) y un vehículo que puede realizar las dos últimas (2 combinaciones). A diferencia del caso anterior, pondremos las diez mejores soluciones presentadas en la tabla 5.14

Número	Solución	Altitud	Distancia	Combustible
10	0.492962	0.9	0.734041	0.830769
15	0.502624	0.9	0.782351	0.830769
28	0.503049	0.91	0.663364	0.94188
29	0.504859	0.91	0.672417	0.94188
31	0.504873	0.9	0.793595	0.830769
33	0.512711	0.91	0.711674	0.94188
34	0.51041	0.94	0.699228	0.912821
35	0.51222	0.94	0.708281	0.912821
42	0.448442	0.96	0.509562	0.77265
51	0.50247	0.94	0.659529	0.912821

Tabla 5.14: Soluciones Prueba 5

La mejor solución de asignación es la número 33 cuyas asignaciones están en la siguiente tabla (5.15):

Asignación
V8 realiza T1
V9 realiza T2
V8 realiza T3
V8 realiza T4
V9 realiza T5

Tabla 5.15: Asignaciones de la mejor solución - Prueba5

Como se puede comprobar la solución es muy parecida al caso anterior, pero hay pequeñas diferencias. El gasto de gasolina es mayor ya que la distancia recorrida en este caso es mayor por todos los vehículos, en consecuencia, el valor final tanto de gasolina como de distancia sean menores que en el caso anterior, a pesar de que realizan mas tareas entre los dos vehículos.

6. G1, G2 realiza M1, M2
   Vamos a aumentar la dificultad de las pruebas, obligando a los diferentes conjuntos de vehículos realizar las dos misiones a la vez. Empezaremos por G1 y G2 (con la nueva altitud adjudicada en la prueba 3), aunque G1 no pueda realizar ninguna tarea, afectará al tiempo de ejecución de la prueba.

   Como hemos visto anteriormente, G2 al realizar la M1 hay 32 posibles soluciones y con M2 habrá 2 soluciones, por lo que en este caso, obtendremos 64 posibles soluciones. Al igual que anteriormente, mostraremos las diez mejores soluciones en la tabla 5.16.

Número	Solución	Altitud	Distancia	Combustible
31	0.230602	0.866667	0.817691	0.621667
32	0.228641	0.875556	0.834192	0.576667
47	0.230521	0.866667	0.816877	0.621667
48	0.22866	0.875556	0.834373	0.576667
55	0.230327	0.866667	0.81494	0.621667
56	0.228568	0.875556	0.833455	0.576667
59	0.230022	0.866667	0.811887	0.621667
61	0.229605	0.866667	0.80772	0.621667
63	0.232082	0.853333	0.812488	0.655
64	0.229435	0.862222	0.822132	0.61

Tabla 5.16: Soluciones Prueba 6

La mejor solución es la 63, de la cúal podemos destacar que sus tres valores por separado ninguno es el mejor, pero si que se caracterizan por estar muy cerca del más óptimo o ser el más óptimo con otro, por lo que en definitiva su salida será la mejor en comparación. En la tabla 5.17 podemos ver las asignaciones realizadas.

Asignación
V6 realiza T1
V6 realiza T2
V6 realiza T3
V6 realiza T4
V6 realiza T5
V4 realiza T6
V4 realiza T7
V4 realiza T8
V4 realiza T9
V5 realiza T10

Tabla 5.17: Asignaciones de la mejor solución - Prueba6

7. G1, G3 realiza M1, M2
   Al igual que en el caso anterior, ahora va a ser el grupo tres de vehículos quien va a realizar todas las misiones a la vez. G3 se caracteriza por poder hacer todas las tareas de la M1, la que se obtiene 243 soluciones y en la M2 puede realizar la mayoria de ellas pero no en su totalidad, habiendo un total de 54 soluciones. En consecuencia, en este caso, habra un total de 13122 soluciones (M1xM2). Al igual que en caso anteriores se mostrarán las cinco mejores soluciones en la tabla 5.18.

Número	Solución	Altitud	Distancia	Combustible
6567	0.272206	0.88	0.871119	0.97094
6581	0.272102	0.895	0.869613	0.95641
6585	0.27279	0.895	0.876488	0.95641
6586	0.272571	0.91	0.873829	0.94188
6587	0.272729	0.91	0.875414	0.94188
7054	0.271414	0.91	0.849257	0.94188

Tabla 5.18: Soluciones Prueba 7

De la mejor solución se puede destacar la misión uno la realiza el vehículo ocho en totalidad, esto

es así porque es el más cercano está de todas las posiciones y su gasto de combustible es muy parecido al de los demás. Mientras que la misión dos la realizan entre el vehículo ocho y el nueve, como podemos ver en la tabla 5.19.

Asignación
V8 realiza T1
V8 realiza T2
V8 realiza T3
V8 realiza T4
V8 realiza T5
V9 realiza T6
V9 realiza T7
V8 realiza T8
V8 realiza T9
V9 realiza T10

Tabla 5.19: Asignaciones de la mejor solución - Prueba7

8. G1, G2, G3 realiza M1, M2
   La última prueba que vamos a realizar es todos los vehículos disponibles con todos las misiones. En este caso la cantidad de soluciones se dispara aún más que los demás casos. Debido a que tenemos un total de 839808 soluciones.

   Ya que si desglosas al final acabarían siendo las combinaciones de G2 con M1-M2 y por otro lado las de G3 con M1-M2. De esta manera llegamos a tener un total de 13122 x 64 soluciones. Su cantidad de soluciones fue tan alta que fue inviable sacar una solución clara de tal cantidad de datos, ademas destacar que su tiempo de ejecución fue superior a las tres horas.

9. Pruebas computacionales

   Una vez realizadas las pruebas de análisis de resultados, vamos a comprobar el rendimiento del planificador mediante pruebas computacionales. El principal objetivo de esta prueba es comprobar el tiempo de ejecución y el número de salidas de nuestro planificador. Realizaremos dicha prueba, cogiendo el grupo tres de vehículos y aumentando el número de tareas que puede realizar. Partiremos de la suposición de que todas las tareas de la misión que se le asignen a éstos vehículos las podrán realizar siempre. De esta manera mediante las imágenes 5.1 y 5.2 expondremos las soluciones obtenidas.

   Como podemos ver en la imagen 5.1. Al realizar una búsqueda de tipo DFS, el número de salidas va aumentando secuencialmente en función de la cantidad de tareas, llegando a tener una cantidad altísima de soluciones. Se puede comprobar como en este caso se va triplicando el número de soluciones, esto es así ya que tenemos tres vehículos, en el caso de que tuvieramos cuatro, la cantidad de soluciones se iría cuatriplicando y así sucesivamente en función de los vehículos.

   A su vez, si comparamos con la imagen 5.2 podemos comprobar como el tiempo de ejecución aumenta también considerablemente sobre todo a partir de ejcutar 8 tareas donde el tiempo de ejecución se dispara, hasta llegar a hora y media de tardanza en ejecutar diez tareas. En consecuencia, pudiendo confirmar que la utilización de un algoritmo de búsqeda de tipo DFS comprueba una cantidad de casos demasiado alta al recorrer todas las ramas posibles.

   Por otro lado, si realizamos la misma prueba pero manteniendo fijo el número de tareas de la misión dos y aumentando la cantidad de UAVs que pueden resolver dicha tarea. Podemos afirmar mediante la imagen 5.3 que el número de soluciones aumenta aunque no tan rápido como en los casos anterior debido a que el número de combinaciones posibles a encontrar por DFS no será tan alto como en el caso anterior.

Imagen 5.1: Soluciones en función de las tareas

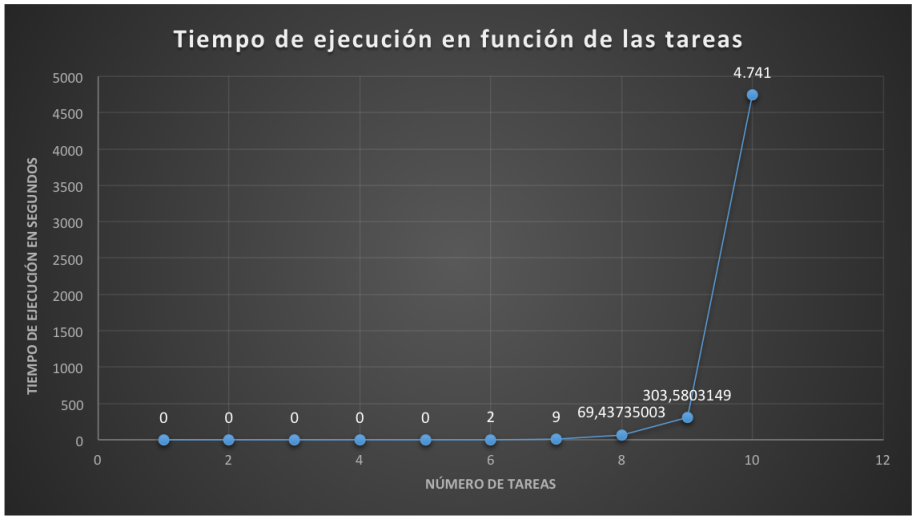


Imagen 5.2: Tiempo de ejecución en función de las tareas

En la tabla 5.4 se recogen todos los tiempos de ejecución de las nueve pruebas realizadas, comprobando como la realización de la última prueba llevo la cantidad de más de tres horas para un caso no tan complejo como deberían ser nueve vehículos y diez tareas.

En la siguiente sección expondremos a las conclusiones que hemos llegado con la realización de todas estas pruebas.

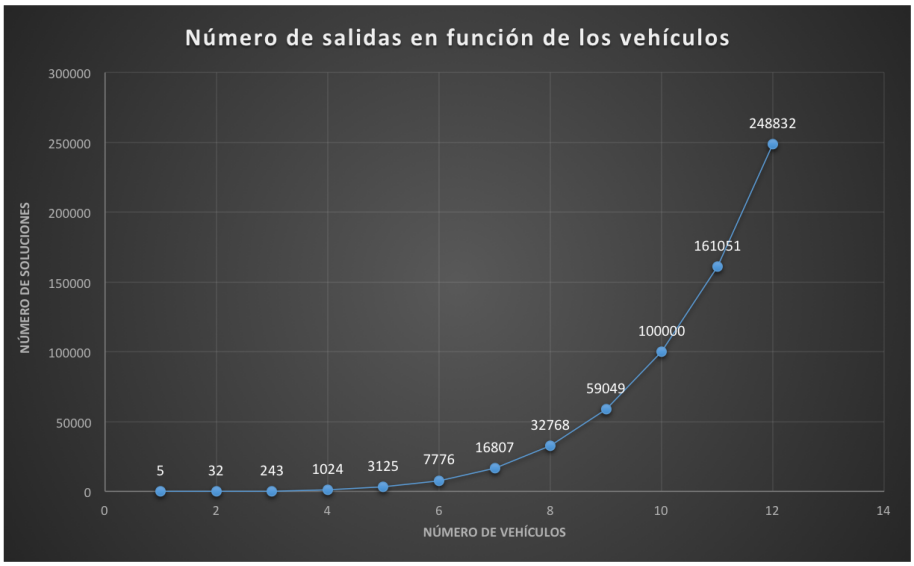


Imagen 5.3: Soluciones en función de los vehículos

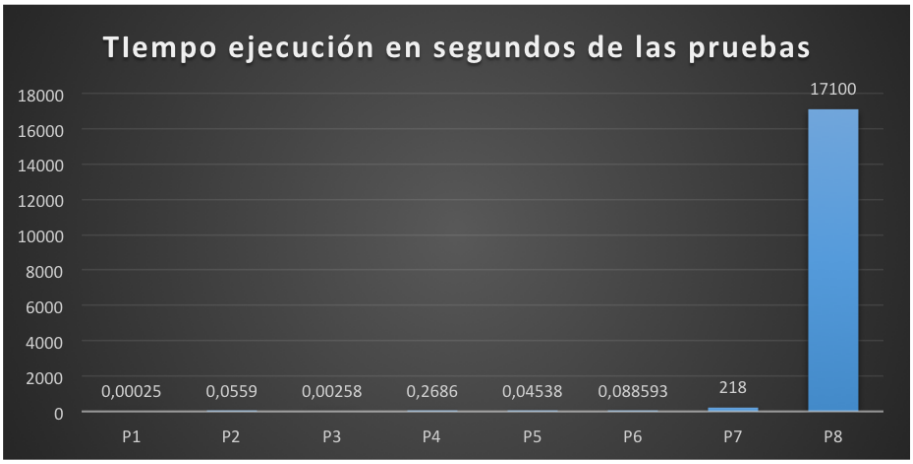


Imagen 5.4: Soluciones en función de los vehículos

6

Conclusiones y trabajos futuros

6.1. Conclusiones

Implementando nuestro planificador basado en CSP se pretendía realizar un conjunto de asignaciones de forma óptima en función de un grupo de restricciones basadas en un entorno lo más real posible. En nuestro caso, se utilizó un algoritmo de búsqueda de tipo DFS para ver como respondía ante la necesidad de comprobar todas las asignaciones.

Primero analizando el conjunto de salidas obtenidas en general, se puede decir que la distancia es el factor más relevante. Aunque en el caso de que la distancia hasta las posiciones sea corta y el gasto de combusitble del vehículo sea bajo, este vehículo suele ser el encargado de realizar las tareas, teniendo en general una salida no muy alta. Hemos podido comprobar como la mejor salida de la tabla 5.14 es la mayor de todas, ese caso es el mejor porque en dicha prueba los vehículos son los que más cerca están de las posiciones y la distancia que recorren es menor que en las demás pruebas. Todos estos datos de salida, variarían mucho en función de como normalizar los factores seleccionados.

Inicialmente cuando usamos una cantidad muy pequña de datos su rendimiento es bueno, pero en el momento que se aumenta el número de vehículos-tareas, tanto el número de soluciones como tiempo de ejecución se empiezan a disparar. Es verdad que en las imágenes 5.1 y 5.2 los vehículos supuestos no encuentran restricciones a la hora de realizar la misión, pero dicho caso también puede ocurrir en una situación real. Por otro lado, como podemos ver en la imagen 5.4 cuando realizamos la última prueba con nueve vehículos y diez tareas con sus respectivas restricciones el rendimiento es el peor, tardando un tiempo altísimo.

Visto todo lo expuesto anteriormente y analizando los resultados se puede llegar a la conclusión que un planificador de misiones basado en restricciones, mediante un algoritmo de busqueda en profundidad (DFS) no es óptimo. En consecuencia también se puede concluir que el algoritmo de búsqueda utilizado tras la asignación de los vehículos es realmente importante, la utilización de otros algoritmos de búsqueda como algoritmos de poda deberían mejorar considerablemente el rendimiento del planificador.

Como futuras mejoras se puede considerar la utilización de un algoritmo de búsqueda de tipo Branch and Bound, que se caracterizará por podar y no recorrer todas las ramas del árbol de asignaciones, además de la creación de una función de evaluación analizando cuales serían los pesos indicados para los factores elegidos anteriormente o para nuevos factores, obteniendo así el resultado más óptimo.

6.2. Trabajos futuros

Al inicio del trabajo, se realizó una investigación sobre simuladores de misiones para UAVs, un trabajo futuro puede ser la implementación de un planificador de misiones mediante restricciones que se pueda simular en alguna aplicación de éste tipo. A continuación se hablará sobre un simulador el cuál nos pareció realmente útil y fácilmente manejable.

6.2.1. Estudio de herramientas para la simulación de planes para UAVs

Actualmente, las aplicaciones para simular o que tienen relación con los UAV son muy pocas, o están en desarrollo. Estas aplicaciones se utilizan para simular posibles misiones con ciertas características específicas. Pudiendo utilizar todo tipo de vehículos así como poder añadir cualquier tipo de sensor.

Tras una búsqueda de este tipo de aplicaciones nos encontramos con que muy pocas te permiten introducir restricciones o características por el estilo, y que la única que te admitía una gran cantidad de opciones era STK.

Está aplicación te daba la opción de simular todo tipo de vehículos en toda la tierra, así como la utilización de satélites o la introducción de restricciones en los vehículos. Hay dos versiones una gratis, y otra de pago y las dos solo se pueden utilizar en Windows (XP). Antes de simular una misión y así comprobar el funcionamiento de la aplicación vamos a ver que debe o suele contener una misión con UAV:

- Escenario: Por escenario podemos entender un ciudad, una coordenada, una dirección de un pueblo e incluso un puerto marítimo. Así como un territorio más extenso como puede ser el golfo de México. El programa te deja ver la tierra en formato 2D o 3D.
- Posición: Posición del mapa o territorio por el cual debe pasar nuestro vehículo para realizar una misión específica.
- Vehículo: No tiene por que ser solo de tierra, si no también puede ser de aire y mar, como helicóptero o barco. Pero también se puede llegar a utilizar un satélite para que envíe órdenes a alguno de los vehículos.
- Sensor: Tal como su nombre indica son los diferentes sensores que podemos añadir a los vehículos, como pueden ser una cámara óptica o una de infrarrojos. En el anexo B podemos ver un sensor que actualmente se está utilizando y sus características específicas.
- Restricciones: Las restricciones pueden ser desde zonas por las que no puede pasar el vehículo así como el límite de llegada del sensor, o la altitud del vehículo de aire.

Visto lo mínimo que debe contener una misión con UAV, en la sección del simulador (4.2.2) podemos ver como se realiza una misión de ejemplo y ver los datos que podemos sacar de una misión.

6.2.2. Simulador, STK-AGI

Abriremos el programa STK y creamos un proyecto.

Lo primero que deberemos hacer es crear o elegir un escenario, en este caso, vamos a utilizar un escenario proporcionado en la base de datos, el Golfo de México que podemos ver en la imagen 6.1.

Imagen 6.1: Realización o creación de los objetivos

Acto seguido vamos a introducir los objetivos o targets, por los que deberá pasar nuestro vehículo. En este caso vamos a crear hasta 7 objetivos y una zona aeroespacial prohibida que se llamará Restricted_Airspace y veremos en 6.2.

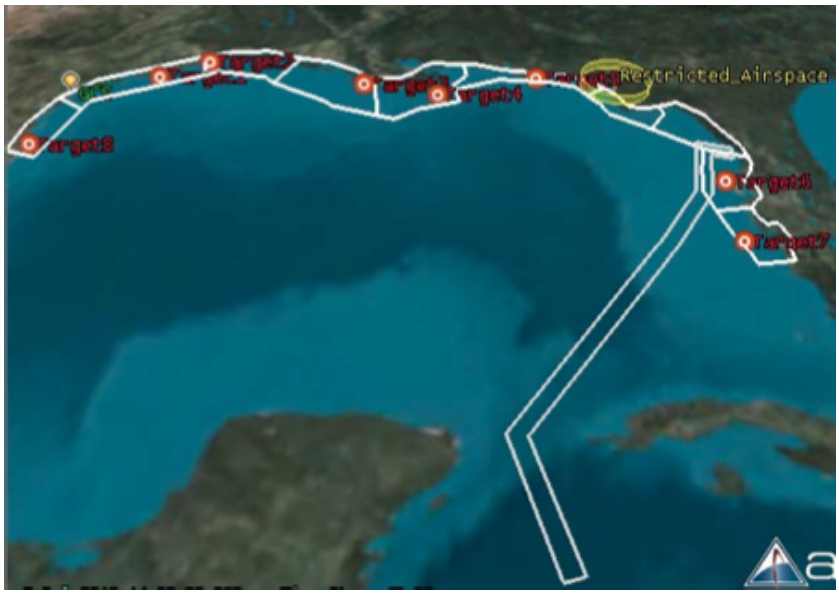


Imagen 6.2: Realización o creación de los objetivos

Después, introducimos los vehículos correspondientes, en este caso, un avión. El programa te da la opción de introducir muchos tipos de aviones.

Una vez elegido el vehículo en cuestión para la misión, deberemos elegir la ruta a llevar por éste. Para realizar esto introduciremos varios objetivos (o puntos del camino) que será por donde pase nuestro vehículo. Estos objetivos se introducen con las coordenadas de cada punto exacto o eligiendo la zona

del mapa. A su vez también podemos elegir la altitud por la que queremos que pase nuestro vehículos por esa área exactamente, todo este proceso lo se puede visualizar en 6.3.

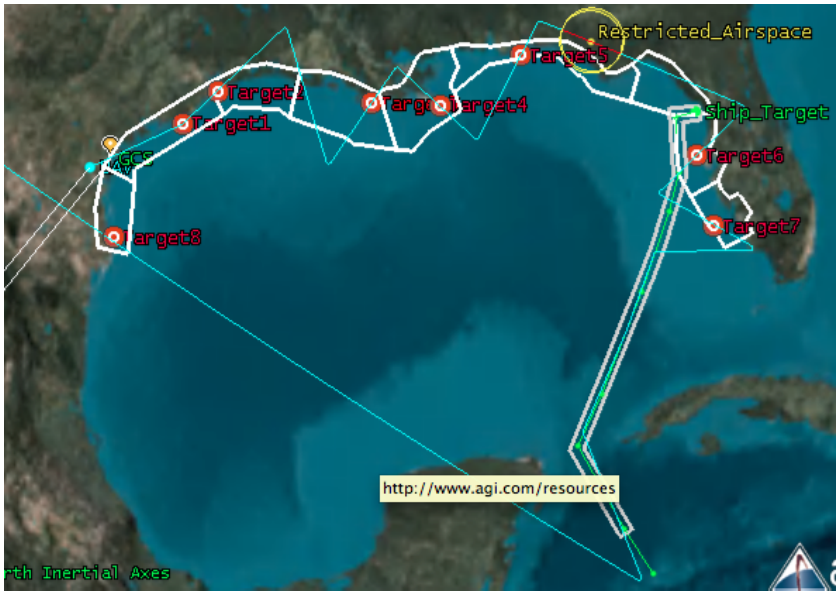


Imagen 6.3: Definicioón de un área restringida

Cuando ya tenemos la ruta de nuestro vehículo así como los objetivos principales, solo queda añadir los sensores o sensor al vehículo. En este caso el sensor que vamos a introducir tiene las siguientes características:

- Sensor de tipo cónico
- Ángulo de 70 grados

Haciendo zoom en el vehículo podemos ver un ejemplo de como sería el vehículo en la imagen 6.4.

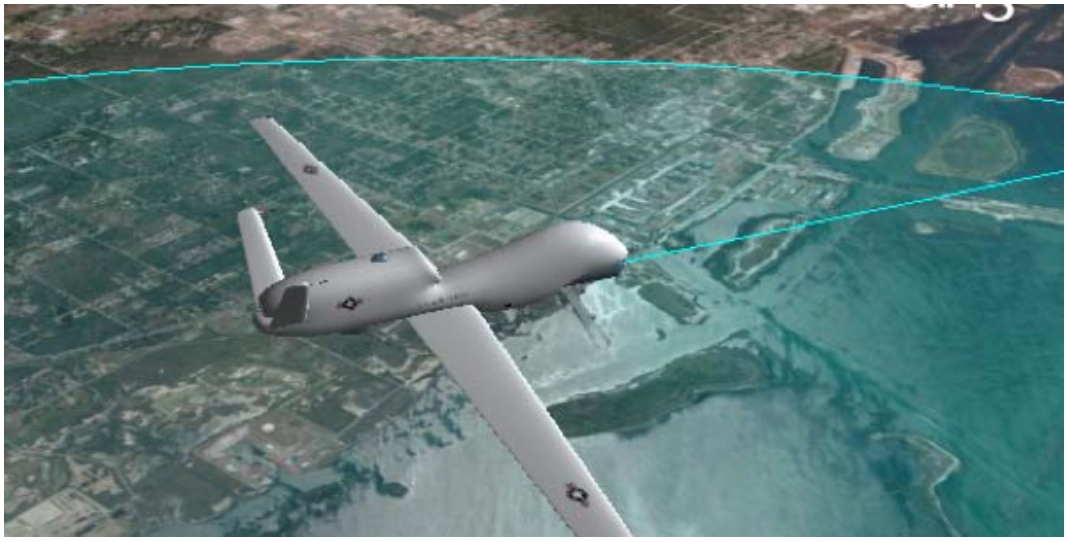


Imagen 6.4: Visualización de un vehículo mediante zoom

Una vez introducidos todos los datos correspondientes ya podemos empezar a simular nuestro proyecto y las conclusiones que sacamos con los datos de salida. Pulsando en el botón derecho del sensor

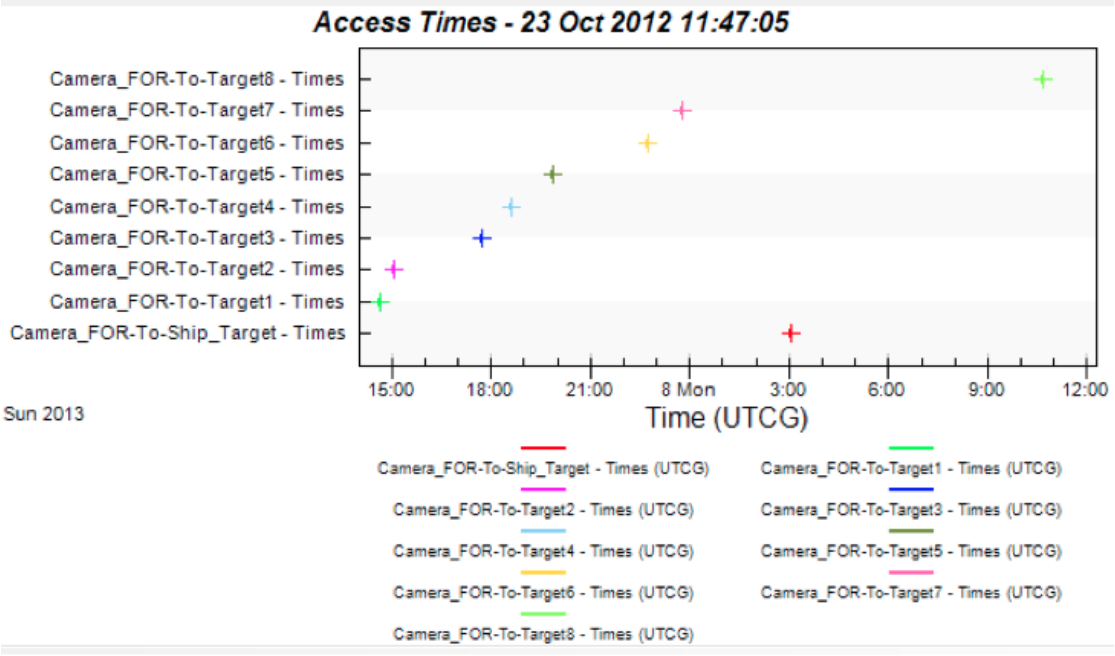


Imagen 6.5: Datos de salida 1

AreaTarget-Restricted_Airspace-To-Aircraft-UAV: Access Summary Report

Restricted_Airspace-To-UAV

	Access	Start Time (UTCG)	Stop Time (UTCG)	Duration (sec)
	1	7 Jul 2013 20:23:17.383	7 Jul 2013 20:55:41.445	1944.062
Global Statistics				
Min Duration	1	7 Jul 2013 20:23:17.383	7 Jul 2013 20:55:41.445	1944.062
Max Duration	1	7 Jul 2013 20:23:17.383	7 Jul 2013 20:55:41.445	1944.062
Mean Duration				1944.062
Total Duration				1944.062

Imagen 6.6: Datos de salida 2

podemos ver diferentes resultados, como el tiempo que ha pasado por cada objetivo o las zonas restringidas. A continuación dos imágenes de estos dos casos, 6.5, 6.6:

De esta manera podemos ver los diferentes datos que hemos sacado sobre la duración del trayecto, así como las horas exactas.

Glosario de acrónimos

- **UAV**: Unmanned Air Vehicule
- **UAS**: Unmanned Air System
- **RUAS**: Rotor Unmanned Air System
- **CSP**: Constraint satisfaction problem
- **GNC**: Guidance, navigation, control system - Sistemas de Guía, Navegación, Control
- **HSA**: Algoritmos heurísticos de búsqueda
- **PPU**: Planificación bajo incertidumbre
- **MC**: Complejidad de la misión
- **EC**: Complejidad del entorno
- **ESI**: Independencia del sistema
- **NASA**: Administración Nacional de la Aeronáutica y del Espacio
- **DFS**: Depth-first search, búsqueda en profundidad

Bibliografía

[1] Mayer Schwartz Norman Delisle. *A programming environment for CSP*, volume 22. 1987.

[2] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. page 64, Marzo 2011.

[3] B. T. Clough. Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway? In *Proceedings of the Performance Metrics for Intelligent Systems (PerMIS)*, 2002.

[4] T Merz. Building a system for autonomous aerial robotics research. In 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles, 2004.

[5] P. Pettersson, P. Doherty. Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle. pages pp. 1–6, Canada, June 2004. In Proceedings of the ICAPS-04 Workshop on Connecting Planning Theory with Practice,.

[6] Mettler. Goerzen, Kong. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, pages 57, 65–100., 2010.

[7] Takahashi Howlett, Whalley. A system for 3d autonomous rotorcraft navigation in urban environments. Honolulu, August 2008. In Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit, AIAA-2008- 7412.

[8] Roy. He, Bachrach. Efficient planning under uncertainty for a target-tracking micro aerial vehicle. In Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, AK., May 2010.

[9] Kendoul. Merz. Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter. San Francisco, CA, September 2011. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.

[10] Doherty Conte. Vision-based unmanned aerial vehicle navigation using geo-referenced information. *EURASIP Journal on Advances in Signal Processing*, pages 1–18, 2009(10).

[11] Robert Harris Michael Freed, Eric Dahlman. *Apex - Reference Manual 2.4.2*. NASA, nasa ames research center edition, Mayo 2011.

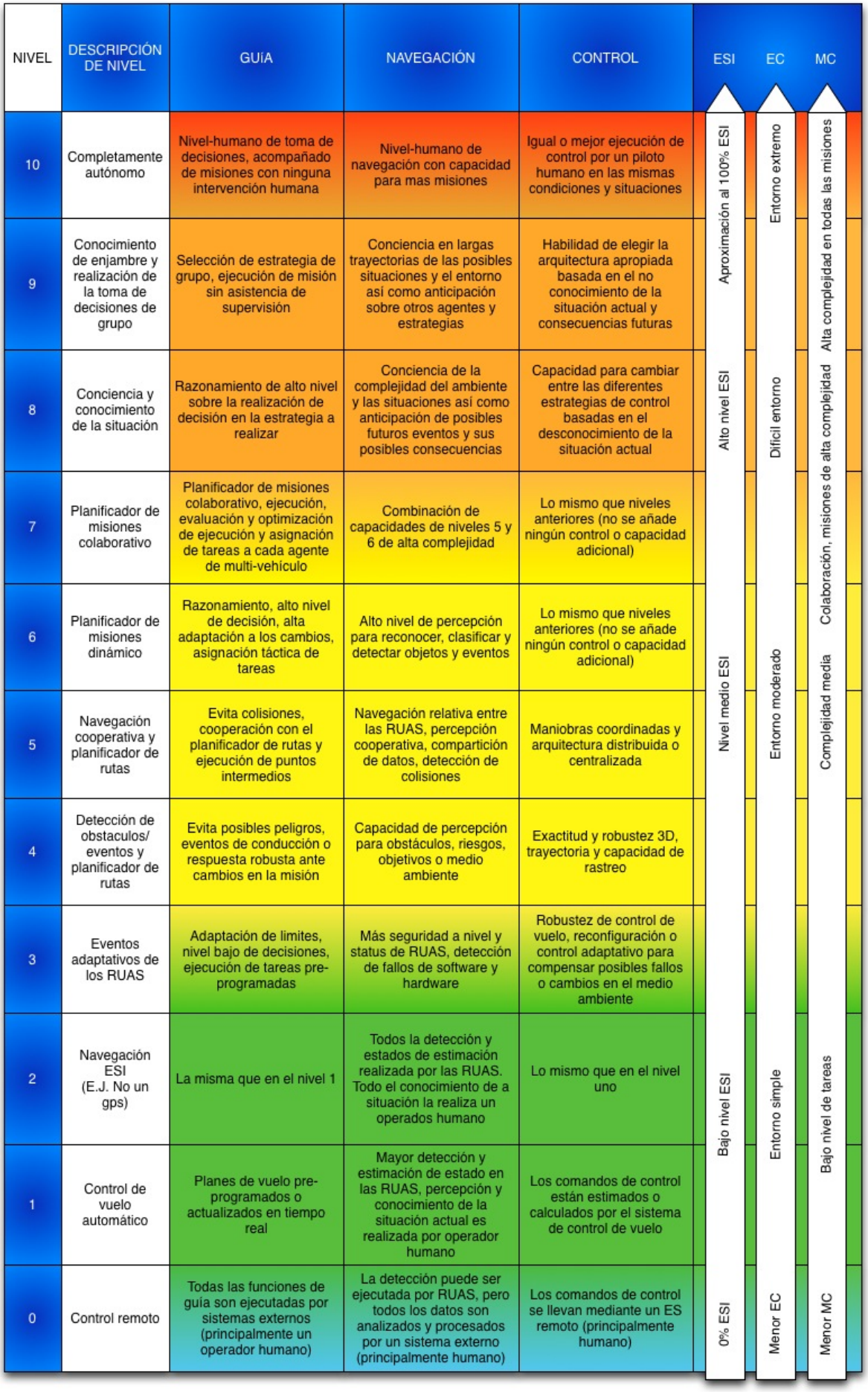
[12] Roland E. Weibel. *Safety Considerations for Operation of Unmanned Aerial Unmanned Aerial Vehicles in the National Airspace System*, pages 39–42. MASSACHUSETTS INSTITUTE OF TECHNOLOGY (MIT), June 2005.

[13] Página con información de librerias de restricciones, http://www.constraintsolving.com/solvers/constraint-libraries.

[14] Página principal de gecode, http://www.gecode.org.

A

Tabla ALFURS



NIVEL	DESCRIPCIÓN DE NIVEL	GUíA	NAVEGACIÓN	CONTROL
10	Completamente autónomo	Nivel-humano de toma de decisiones, acompañado de misiones con ninguna intervención humana	Nivel-humano de navegación con capacidad para mas misiones	Igual o mejor ejecución de control por un piloto humano en las mismas condiciones y situaciones
9	Conocimiento de enjambre y realización de la toma de decisiones de grupo	Selección de estrategia de grupo, ejecución de misión sin asistencia de supervisión	Conciencia en largas trayectorias de las posibles situaciones y el entorno así como anticipación sobre otros agentes y estrategias	Habilidad de elegir la arquitectura apropiada basada en el no conocimiento de la situación actual y consecuencias futuras
8	Conciencia y conocimiento de la situación	Razonamiento de alto nivel sobre la realización de decisión en la estrategia a realizar	Conciencia de la complejidad del ambiente y las situaciones así como anticipación de posibles futuros eventos y sus posibles consecuencias	Capacidad para cambiar entre las diferentes estrategias de control basadas en el desconocimiento de la situación actual
7	Planificador de misiones colaborativo	Planificador de misiones colaborativo, ejecución, evaluación y optimización de ejecución y asignación de tareas a cada agente de multi-vehículo	Combinación de capacidades de niveles 5 y 6 de alta complejidad	Lo mismo que niveles anteriores (no se añade ningún control o capacidad adicional)
6	Planificador de misiones dinámico	Razonamiento, alto nivel de decisión, alta adaptación a los cambios, asignación táctica de tareas	Alto nivel de percepción para reconocer, clasificar y detectar objetos y eventos	Lo mismo que niveles anteriores (no se añade ningún control o capacidad adicional)
5	Navegación cooperativa y planificador de rutas	Evita colisiones, cooperación con el planificador de rutas y ejecución de puntos intermedios	Navegación relativa entre las RUAS, percepción cooperativa, compartición de datos, detección de colisiones	Maniobras coordinadas y arquitectura distribuida o centralizada
4	Detección de obstaculos/ eventos y planificador de rutas	Evita posibles peligros, eventos de conducción o respuesta robusta ante cambios en la misión	Capacidad de percepción para obstáculos, riesgos, objetivos o medio ambiente	Exactitud y robustez 3D, trayectoria y capacidad de rastreo
3	Eventos adaptativos de los RUAS	Adaptación de limites, nivel bajo de decisiones, ejecución de tareas pre-programadas	Más seguridad a nivel y status de RUAS, detección de fallos de software y hardware	Robustez de control de vuelo, reconfiguración o control adaptativo para compensar posibles fallos o cambios en el medio ambiente
2	Navegación ESI (E.J. No un gps)	La misma que en el nivel 1	Todos la detección y estados de estimación realizada por las RUAS. Todo el conocimiento de a situación la realiza un operados humano	Lo mismo que en el nivel uno
1	Control de vuelo automático	Planes de vuelo pre-programados o actualizados en tiempo real	Mayor detección y estimación de estado en las RUAS, percepción y conocimiento de la situación actual es realizada por operador humano	Los comandos de control están estimados o calculados por el sistema de control de vuelo
0	Control remoto	Todas las funciones de guía son ejecutadas por sistemas externos (principalmente un operador humano)	La detección puede ser ejecutada por RUAS, pero todos los datos son analizados y procesados por un sistema externo (principalmente humano)	Los comandos de control se llevan mediante un ES remoto (principalmente humano)

Imagen A.1: Tabla ALFURS

B

Sensor actual

El LP 17D y 18D LP eye-safe laser rangefinders están diseñados para su uso en sistemas de observación compactos para aplicaciones aerotransportadas, marítimas y terrestres.

Incorporan el estado del arte, tecnología de diodo-bombeado alojados en paquetes extremadamente compactos que precisan de eficacia en distancias a blancos móviles.

Cortesía de una cámara de "puntería", la alineación integrada con el sistema de observación de acogida es un ejercicio de conveniencia.

Imagen B.1: Imagen del eye-safe laser

Transmitter	LP 17D	LP 18D
Extinction ratio	37 dB[1]	31 dB[1]
Maximum range	20 km	10 km
Measuring rate	2 pps (cont.)	6 pps (cont.)
Range gate	80 m to 20 km	80 m to 10 km
Minimum range	80 m	
Range resolution	1 m	
Range accuracy	±5 m	
Multiple target resolution	30 m	
Multiple target range	logic 1st, 2nd & last target	
Weight	~520 g	
Size (W x H x L)	59 x 41 x 112 (mm)	
External power (VDC)	14–33 VDC	
Operating temperature	–40°C to +70°C	
Storage temperature	–45°C to +71°C	
Coms interface	RS-422, RS-232	
Mech interface	3 point fixation	
Eye safety	Class 1 (IEC 60825-1)	

[1] Target at 500 m, reflection > 85%, visibility 23 km

Imagen B.2: Tabla con sus características